PC-Techknow 8800mkII

テク ノウ

平松達雄・八木良一 共著　システムソフト監修

PCファミリー・テクニカル・ノウハウ集
PC-8800mkIIシリーズ編

システムソフト

ＰＣファミリー・テクニカルノウハウ集

PC-Techknow8800mkII

平松達雄
八木良一 著

システムソフト

はじめに

パソコンも社会に定着し、ワープロとともに身近なOAとして利用されています。ビジネスユースでは16ビットマシンが主流になりましたが、正にパーソナルなパソコンとしては８ビットマシンが大部分です。

PC-8801mkⅡは、８ビットのスタンダードともいえるPC-8801の後継機種として発売されましたが、その最大の特徴はディスクドライブが内蔵されたことでしょう。これによりカセットでは味わえない、数々の長所をもったディスクベースのソフトウェアがますます利用されるようになり、いまや、PC-8801mkⅡ用に発売されるソフトウェアの多くがディスクベースになりました。８ビットマシンのディスク時代の幕を開けたといっても過言ではありません。

本書は、このPC-8801mkⅡをより高度に活用するために、本体だけでなく、周辺機器も含めたシステムの機能を徹底解剖し、内部構造や有用なテクニック、周辺機器の制御方法などの情報をまとめたものです。また、すぐに使えるプログラムも豊富に紹介しています。座右の書として御活用いただき、読者の方々の利用技術向上の一助となれば幸いです。

パソコンがポピュラーになるにつれて、その使われ方も変化し、使いやすく分かりやすいソフトウェアが求められるようになりました。その結果、ソフトウェアはますます高度化・複雑化し、機械語を使用せずに求められる機能を実現させるのはほとんど不可能になってきました。そこで、本文中での機械語の使用はなるべく避ける、というTechknowシリーズの編集方針も変更を余儀なくされ、今回はかなり積極的に機械語を使用しています。このため、内容に難解な部分があると同時に、項目によってレベルにギャップが生じてしまいましたことを御容赦下さい。

なお、本書の執筆は主に平松が行ない、サブシステム部分を八木が担当いたしました。本書の内容は筆者らが独自に調査・解析したものであり、文責は筆者らにあります。また、運用上の影響については責任を負いかねますので御了承下さい。

最後になりましたが、本書を出版するにあたって、日本電気株式会社の関係者の方々、株式会社システムソフトの樺島社長や小金丸技術部長を始めとする編集スタッフの方々には大変お世話になりました。また、栗山浩一・松尾篤也両氏には著書からの大規模な引用を快く了承していただきました。この場を借りて心より御礼申し上げます。

昭和60年４月

平松　達雄　記

編集部より

○本書の内容に関する御質問について

本書は著者の独自の調査・解析に基づくものですので、内容についての電話質問に、編集部または弊社ユーザーサポート部ではお答えしかねる場合がございます。

内容に関するご質問は、正確を期するためにも、書面にてお寄せください。

○アンケート葉書について

システムソフトでは発刊後に発見された誤り等について、バグ情報をお送りしています。本書に添付されているアンケート葉書を御返送いただいた方には、自動的にバグ情報をお送りいたしますので、アンケート葉書は必要事項をご記入の上、送り返してください。

○本書の運用上の影響について、株式会社システムソフトでは責任を負いかねますので、御了承ください。

― 目　次 ―

第1章　システム概説

1-1　ソフトウェア

PC-8801mkⅡにはN88-BASICとN-BASICという2種類のBASICが搭載されています。また model　20,30ではディスクドライブがあるために、他の汎用OSを動かすことができ、その上で各種の高級言語やアプリケーションを動かすことができます。

1）N88-BASIC

PC-8801mkⅡの中心となる言語で、ユーザの多くがこの言語を使われることと思います。本書でもこの言語を念頭において話をすすめていきます。

2）漢字BASIC

PC-8801mkⅡには漢字ROMが標準装備されていますから、これを用いた漢字BASICが用意されています。漢字BASICについては第8章をご覧下さい。

3）N-BASIC

PC-8001で採用されていたBASICで、PC-8801(mkⅡ),PC-8001(mkⅡ)に共通に使えます。機械語およびメモリ配置を考慮しなければPC-9801(E／F)でもオプションで使用できます。

4）CP／M

80系の8ビットマシンでは最もポピュラーな汎用OS(基本ソフト)です。生い立ちが古いために少し使いづらい所もありますが、その上で走るソフトウェアの数は絶大です。CP／Mにはいくつかのバージョンがあり、現在最も普及しているのはVer 2.2です。PC-8801mkⅡ用としては、NECからPS 88-104-2Wが発売されています。これは、ハードディスクやRAMディスクもサポートしていて、豊富なユーティリティーが付属しています。また、CP／MはMSA(㈱マイクロソフトウェアアソシエイツ)などのソフトハウスからも発売されています。Ver 2.2以外のバージョンとしては、MSAなどからCP／M Plus(CP／M Ver 3.0)という上位バージョンが発売されています。

またPCP／Mという、Ver 2.2とVer 3.0との中間的機能を持ち、Ver 2.2の欠点を解決したコンシューマ向けのバージョンもあります。

1-2　ハードウェア

PC-8801mkⅡは、パソコン本体ともいうべきメインシステム部分と、ミニディスクドライブを制御するサブシステム部分とで構成されています。全体のブロック図の概略は次のようになっています。

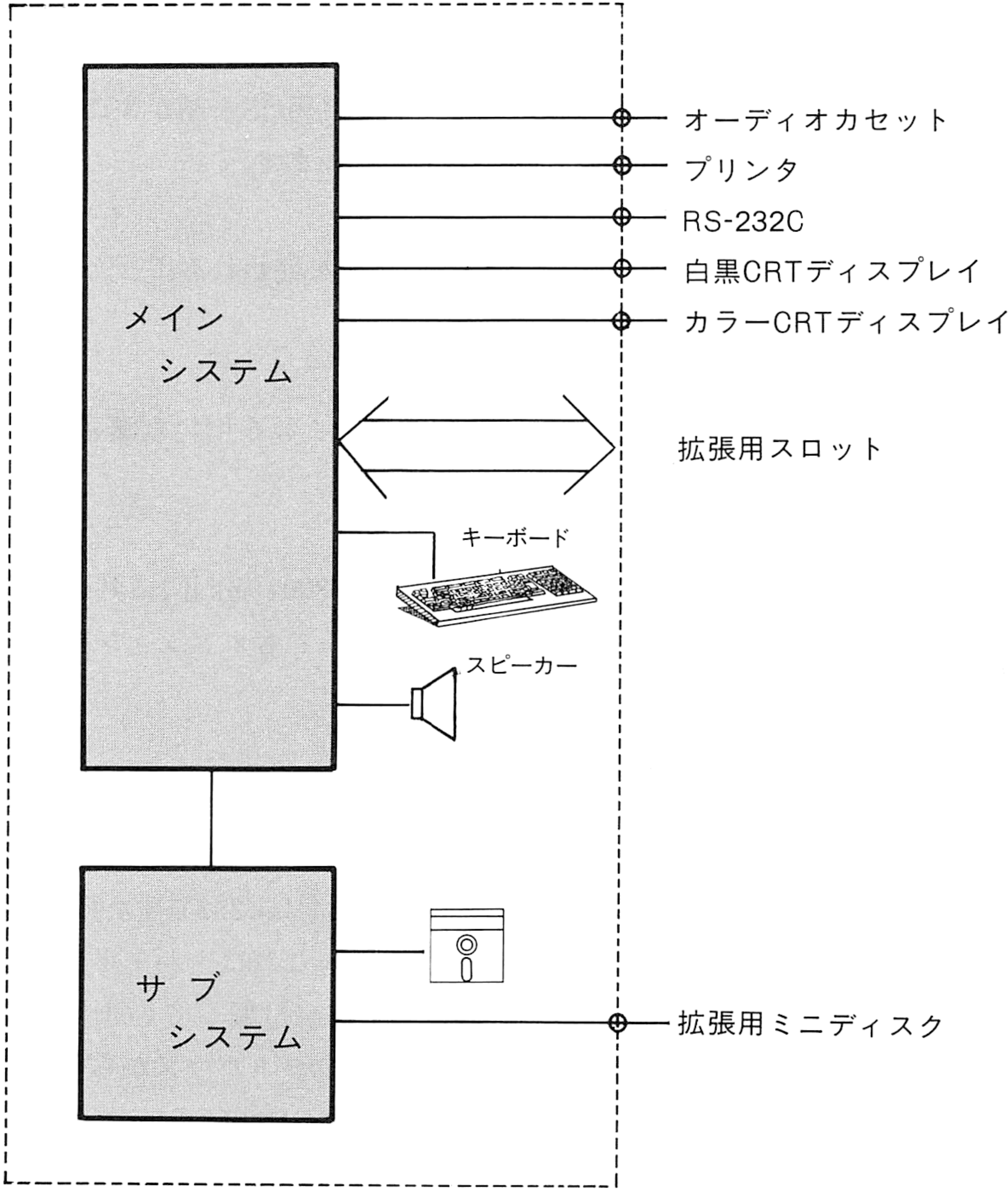



図1-2-A　全体のブロック図

1-3　メインシステムとサブシステム

前節で述べたように、PC-8801mkⅡには2つのCPU(どちらもZ-80)があり、それぞれメインシステム、サブシステムを形成しています。メインシステムは旧PC-8801本体に相当するもので、パソコンとしての働きはほとんどこのメインシステムで行なっています。これに対して、サブシステムはPC-80S31に相当するもので、フロッピィディスクの制御だけを行なっています。

とは言っても、サブシステムも16KバイトのRAM(うちフリーエリア7Kバイト強)を持っていますから、うまく使えばかなりのことができます。

メインシステムとサブシステムのインタフェースには、μPD8255が使われています。

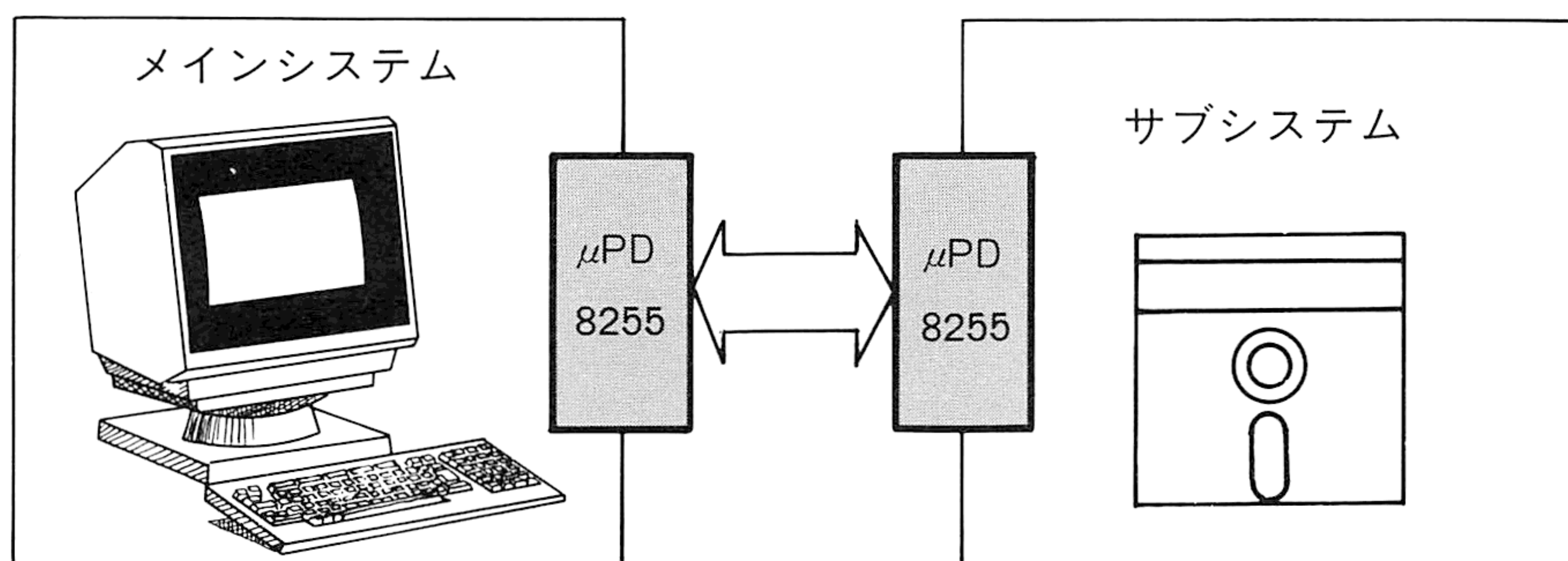



図1-3-A　メインシステムとサブシステム

インタフェースの方法などについては第10章サブシステムを見て下さい。

第2章　メモリモード

メインシステムには標準で112KバイトのRAMと72KバイトのROMが搭載されていますが、これらはいくつかのバンクに分けられ、それぞれの役割が決まっています。この章では主にこれらのバンクの使用方法について述べますが、バンク切換えはBASICを用いて行なうことができませんので、ほとんど機械語を用いた説明になります。

2-1　メモリマップ

メインシステムのメモリは次のような構成になっています。

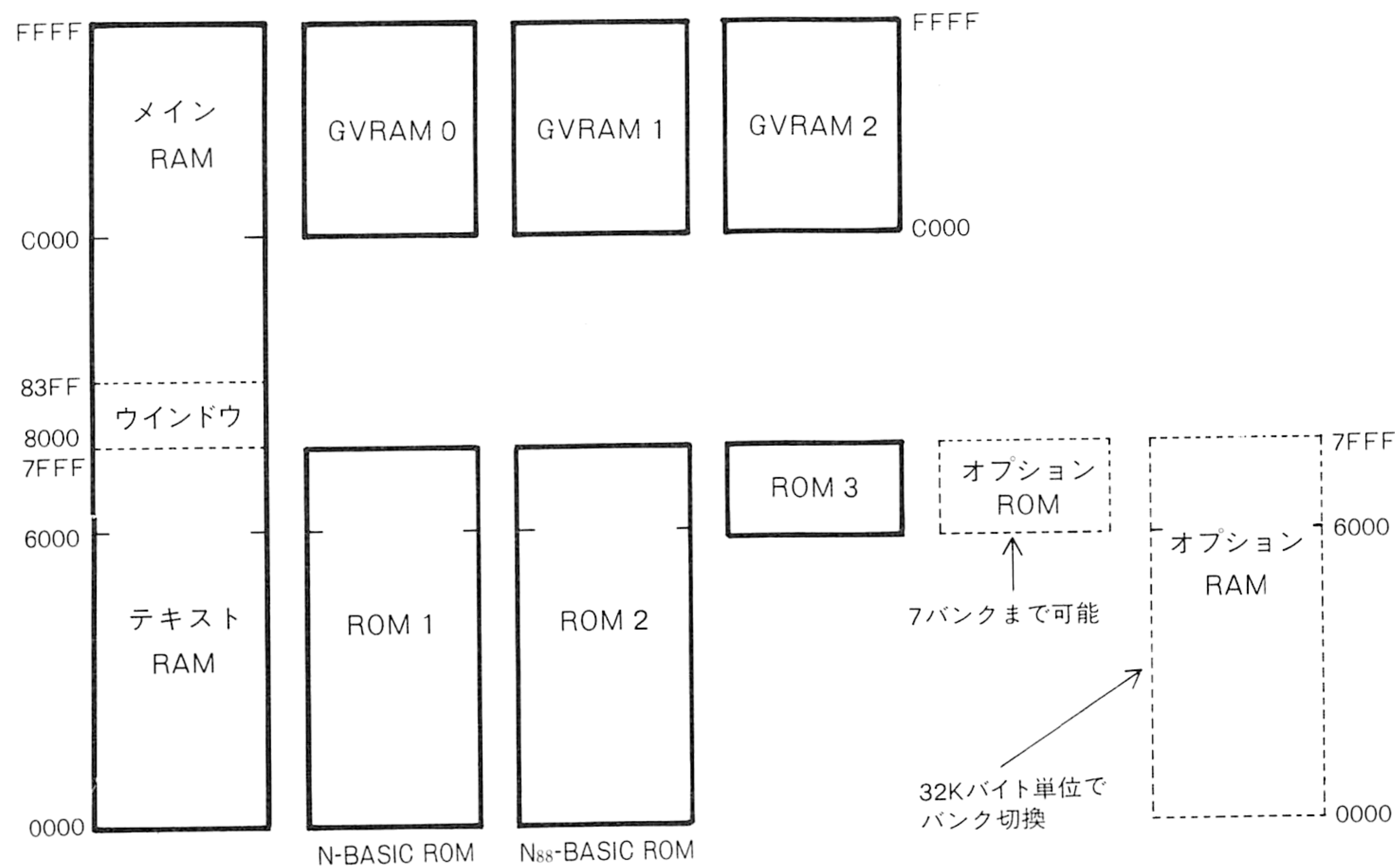



図2-1-A　メインシステムのメモリマップ

1)メインRAM

BASICのワークエリアや変数エリアとして使われます。N-BASICではBASICプログラムもここに置かれます。

2)テキストRAM

N88-BASICのプログラムがここに置かれます。N-BASICモードでは使用されていません。

3)グラフィックRAM

N88-BASICのグラフィック表示用RAMです。光の三原色に対応して3つのバンクがあります。

4)拡張ROM

モード1(後述)のとき、拡張ROMエリアとして6000H～7FFFHまでの8Kバイトを7バンク分使用することができます。

5)拡張RAM

拡張RAMは0000H～7FFFHまでの32Kバイト使用することができます。

2-2　メモリモード

メインシステムのメモリモードとして次の3つのモードがあります。

N-BASICモード　(モード0)

N88-BASICモード(モード1)

64K RAMモード　(モード2)

以上3つのモードはプログラムによって切換えることができます。

(1)モード0

このモードではPC-8001と同じ働きをします。図2-1-AにおけるROM1が選択され、メモリマップは図2-2-Aのようになります。メモリアドレス0000H～7FFFHのRAMは使用できませんが、書き込みだけは常に可能です。このエリアのRAMからデータを読み込んでくるためにはモード2にする必要があります。

また、PC-8001のN-BASICを使用しているプログラムは、このモードで実行できますが、PC-8001の拡張ROM(6000H～7FFFH)をアクセスするようなプログラムは実行できません。

アドレス	内容
FFFF EA00	ワークエリア
E9FF 8000	スタック ストリング用スタック プログラム 変数 エリア
7FFF 0000	ROM 1 (N-BASIC インタプリタ)

図2-2-A　モード0メモリマップ

(2)モード1

このモードでは図2-1-AにおけるROM2、ROM3、が選択されます。N88-BASICはこのモードで動きます。この場合のメモリマップは図2-2-Bとなります。BASICのテキストエリア0000H～7FFFHはCPUのメモリマップ上にはありません。メインRAMのうち、8400H～FFFFHはCPUのメモリマップ上にあります。8000H～83FFHはテキストRAMを読み書きするための「ウィンドウ」となります。これについては後述します。

FFFF
ワーク
変数
VRAM
エリア
8400
TEXT WINDOW
8000
テキスト
RAM
0000

ROM 2
N88-BASIC
インタプリタ

7FFF
ROM 3
6000

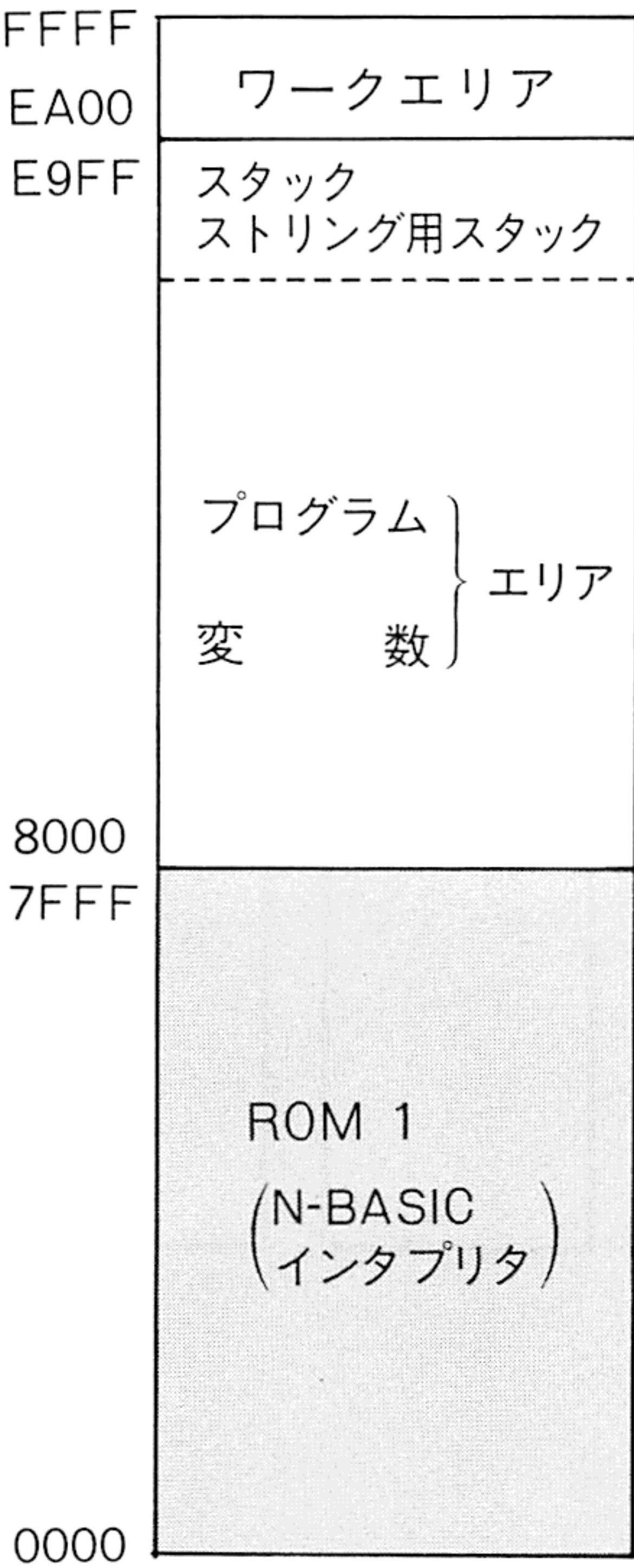
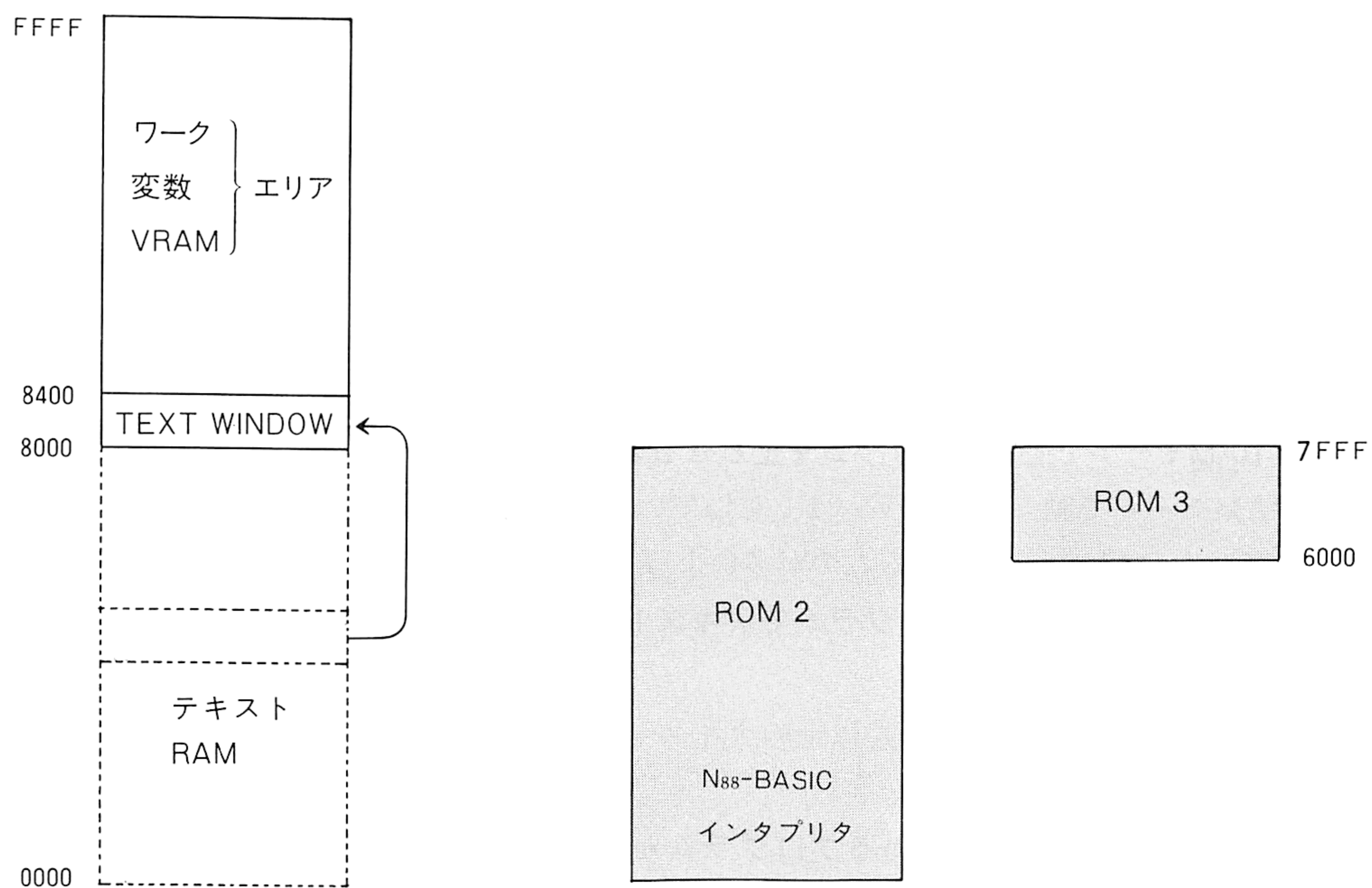



図2-2-B　モード1メモリマップ

(３)モード2

このモードはROM１～３をマスクし、CPUのメモリマップ(64Kバイト)はすべてRAMで構成されます。図２-２-Cにメモリマップを示します。このモードではBASICは動きません。CP／Mその他ほとんどの汎用OSはこのモードで動きます。

FFFF
64Kバイト
RAM
8000
0000

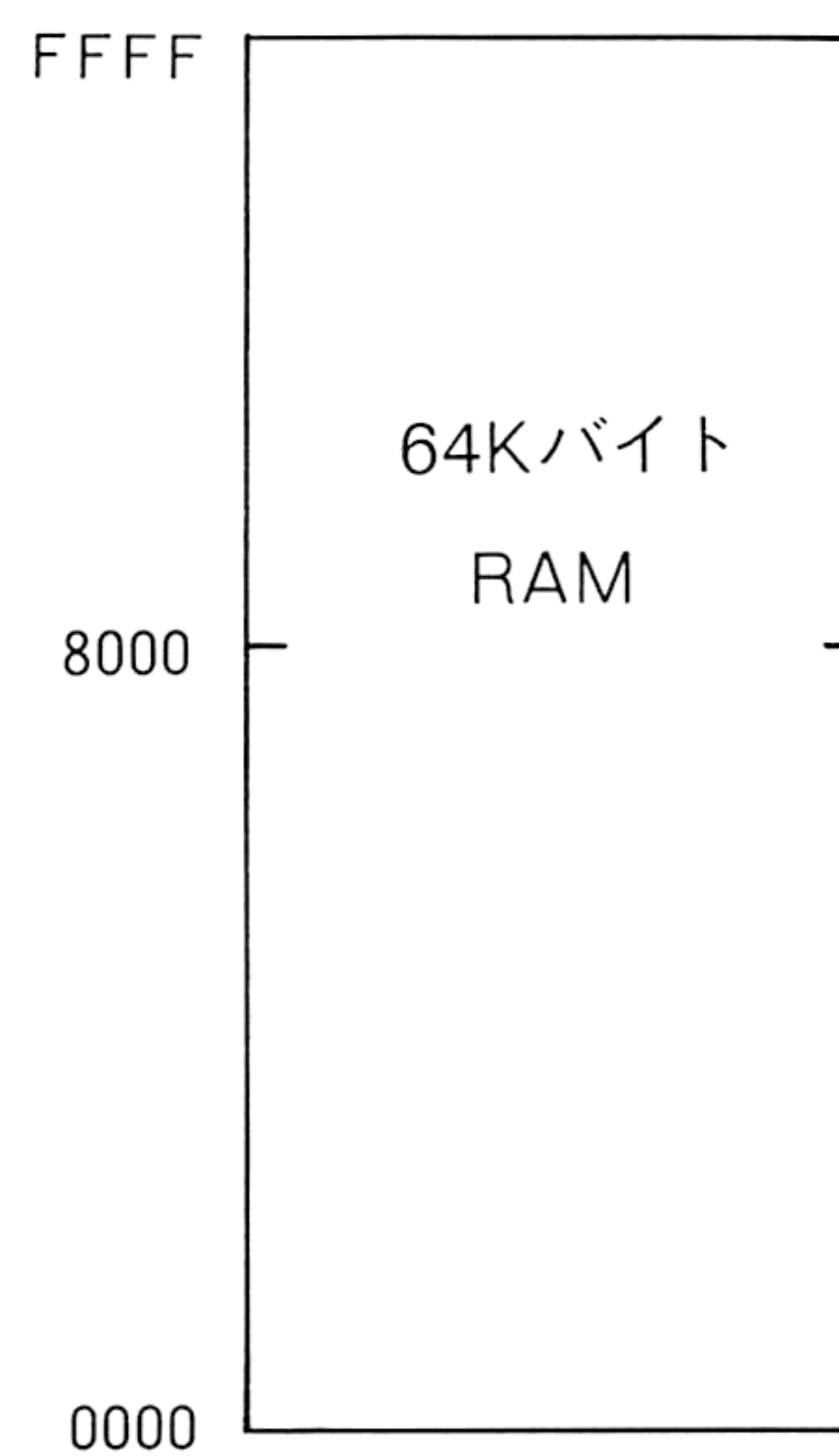



図2-2-C　モード2メモリマップ

2-3　メモリモードの切換え

モード切換えは次のような時に使われます。

(1)ディップスイッチがN-BASICモードになっている時

PC-8801mkIIは、リセット時には必ずモード1(N88-BASICモード)となっています。そしてN88-BASICインタプリタが起動されるのですが、ここでN88-BASICインタプリタはディップスイッチの状態をセンスし、N-BASICモードに設定されていたらモード0に切換えて、N-BASICインタプリタに制御を渡します。

(2)64K RAMモードで動くプログラムを走らせる時

64K RAMモードで動くプログラムは、たいていその最初でモードを2に切換えます。特にIPL中で64K RAMモードにするものが多いようです。

(3)N88-BASICモードで動くプログラムでテキストRAMをウィンドウを通さずにアクセスしたい時

1Kバイト(ウィンドウの大きさ)以上のテキストRAMのエリアを一度にリードしたい時には、このような方法をとります。

(4)64K　RAMモードで動いているプログラムから、N88-BASICまたはN-BASICのROMルーチンを呼び出したい時。

たとえば、BASIC　ROMの中のキー入力ルーチンや1文字出力ルーチンを利用したい時に、一時的にモードを切換えてROMルーチンを呼び出し、もどってきた所でまた64K RAMモードにもどします。この時、呼び出すROMルーチンが使うワークエリアは、あらかじめセットしていなければなりません。実際にはBASICのワークエリア(N88-BASICのときE600H～FFFFH, N-BASICのときEA00H～FFFFH)をそのまま持っておくという方法がとられます。

3つのメモリモードの切換えは、I／Oポート31Hのビット1とビット2で行ないます。

ポート31H(OUTのみ, N88-BASICのワークエリア：E6C2H)

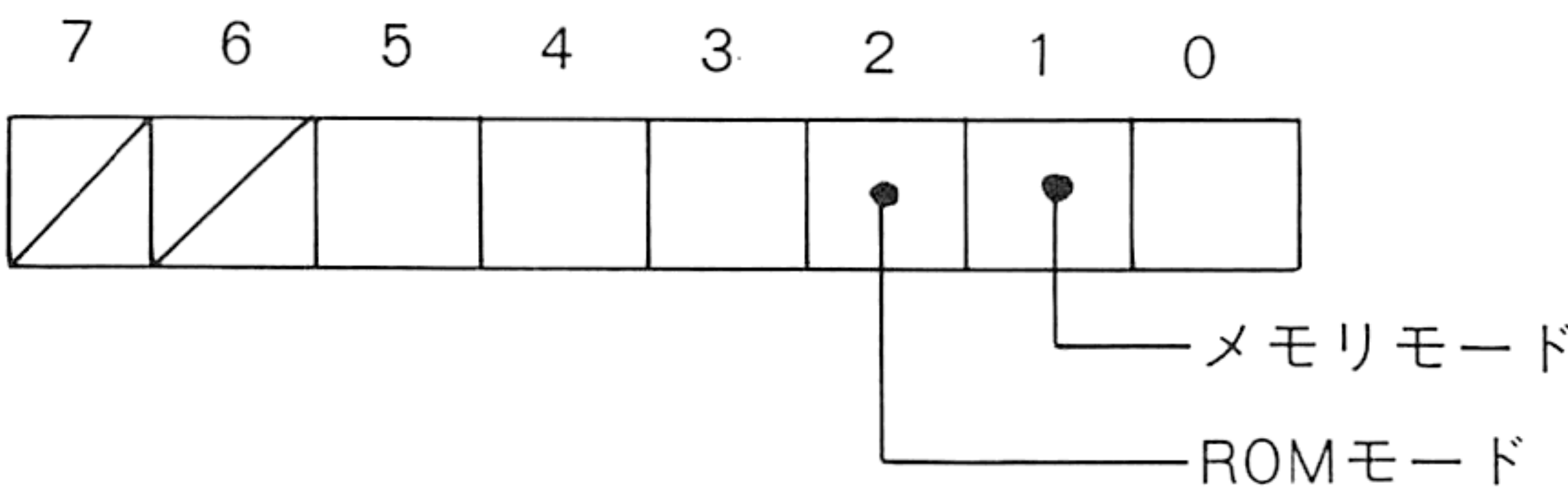
7	6	5	4	3	2	1	0
／	／				ROMモード	メモリモード	

モ　ー　ド	ROMモード	メモリモード
モード　0　(N-BASIC)	1	0
モード　1　(N88-BASIC)	0	0
モード　2　(64K RAM)	—	1

図2-3-A　ポート31Hモード切換え

モードを変更するためにはR MODEとM MODEを前の表のようにセットしたデータをOUTしてやればよいのですが、このとき他のビット(ビット0、ビット3～5)も同時に変更することになります。しかし、他のビットの値を勝手に変えるわけにはいきません。これを避けるためには、現在ポート31Hに出力されているデータを知って、ビット0、ビット3～5はこのデータと同じにしてOUTすればよいわけです。N88-BASICでは、ポート31Hに出力したデータをワークエリアのE6C2H番地に持っています。したがって、N88-BASICのときは次のようにします。

①E6C2H番地の値を読む。

②その値のビット1，2を変更する。

③変更した値をポート31HにOUTする。

④変更した値をE6C2H番地にしまう。

ところで、これをBASICで行なうわけにはいきません。なぜなら、③でモードを切換えると、N88-BASICインタプリタのROMが切換わって、N-BASICのROM(モード0)かRAM(モード1)になってしまうからです。切換わったとたんに暴走してしまいます。(そしてたいていの場合はリセットがかかります。)

そこで、これをアセンブリ言語で書くとこうなります。

モード1→モード2　(N88-BASIC→64K RAM)

```
        DI
        LD      A,(0E6C2H)
        OR      6
        OUT     (31H),A
        LD      (0E6C2H),A
        EI
```

モード2→モード1　(64K RAM→N88-BASIC)

```
        DI
        LD      A,(0E6C2H)
        AND     0F9H
        OUT     (31H),A
        LD      (0E6C2H),A
        EI
```

このプログラムは、最初にDI、最後にEIが入っています。

つまり、これを実行している間は割り込み(NMIを除く)がかからないようにしているわけです。もし、E6C2H番地の値を読んだ後で割り込みがかかり、その割り込みルーチンでE6C2H番地の値を変更したとしますと、割り込みからもどってきた時、アキュムレータに入っている値は古いものとなってしまいます。その値を使ってポート31HにOUTすると、おかしなことになります。そこで、割り込みがかからないようにDIを実行しているのです。

DIではNMIを止めることはできませんが、PC-8801mkⅡでは拡張ボードで使用しないかぎりNMIはかかりません。また、N88-BASICから64K RAMモードにするときは、N88-BASICのインタラプトルーチンがN88-BASIC　ROMを使うので、64K RAMモードの間ずっとDIをかけたままの状態にしておかないと暴走してしまいます。

2-4 ウィンドウ機能

N88-BASICモードの場合、ウィンドウ機能と呼ばれる機能があります。このウィンドウ機能とは、WINDOW文やCONSOLE文のウィンドウとはまったく別のもので、N88-BASICインタプリタの裏側にあるテキストRAMの内容を見る場合に使用されるものです。

N88-BASICモードでは、CPUは0000H～7FFFHのアドレスのメモリを、読み出す場合はROMに、書き込む場合はRAMにアクセスしますので、その範囲にあるテキストRAMの内容を読み出すことができません。これを見るためには、テキストRAMの内容をいったん8000H～83FFHのアドレスに移してから見ます。

これは、テキストRAMの内容をアドレス8000H～83FFHのRAMに書き込んでそれを見ているのではなく、テキストRAMのアドレスをハードウェアで8000H～83FFHに変えて、それを読んでいるのです。また、アドレス8400H～FFFFHまでの31Kバイトのメモリ空間上にあり、直接リード／ライトができますので、ウィンドウを用いる必要はありません。つまり、ウィンドウ機能とは、0000H～83FFHまでのアドレスにあるRAMの内容を読む時に使用するものです。

読みたいアドレスを指定する時に使用するものとして、8ビットのオフセットアドレスレジスタ(OAR)というものがあります。OARはプログラムにより設定できます。まず、OARに8ビットのデータを入れると、そのデータを上位バイト、00を下位バイトとしたアドレスから1Kバイトがウィンドウに映ります。例えば、OARに37Hを入れると、テキストRAMのアドレス3700H～3AFFHまでが映るわけです。3B00H以後を見たいときには、OARをインクリメントして、OARを38Hとし、3800H～3BFFHを映します。

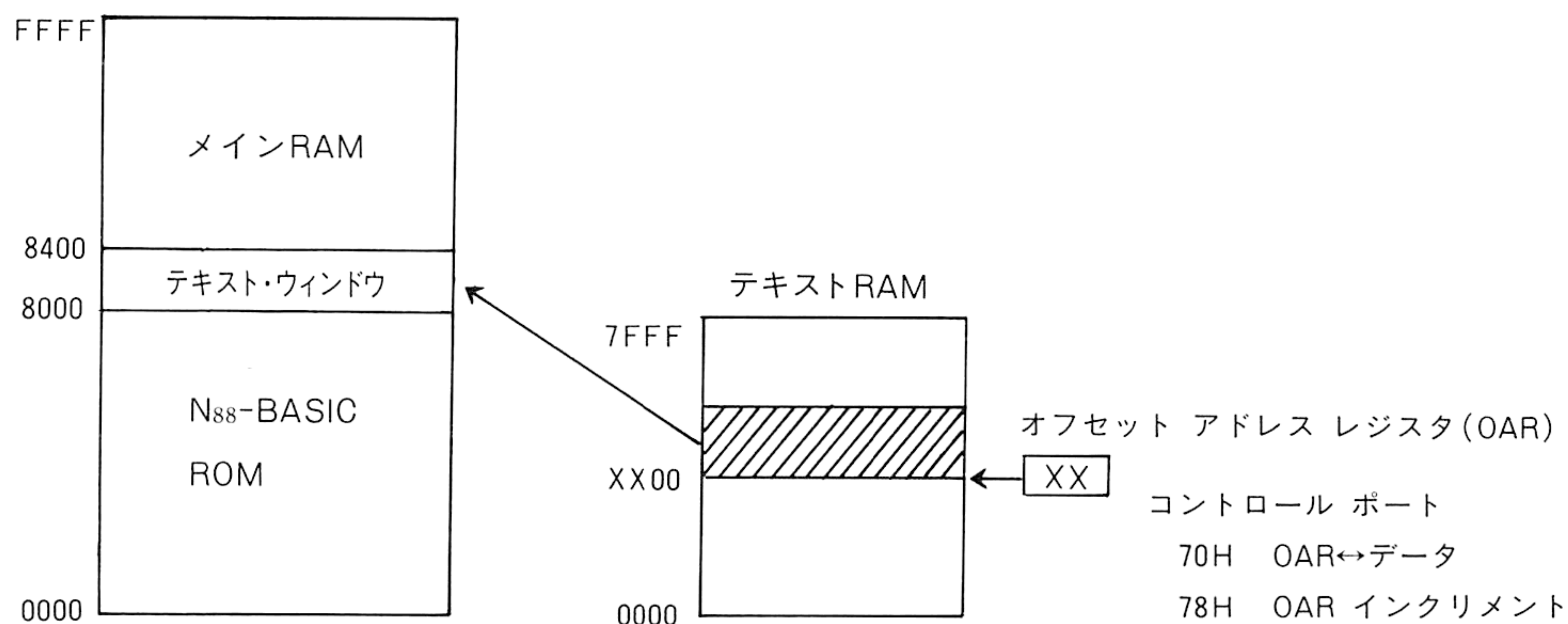



図2-4-A ウィンドウのメモリマップ

OARに関する命令には次の3つがあります。

①OARにオフセットアドレスを設定する	OUT 70H
②OARからオフセットアドレスを読み出す	IN 70H
③OARを＋1する(出力データは意味をもちません)	OUT 78H

それでは実際にやってみましょう。まず次のプログラムを実行して下さい。

リスト2-1

```
10 FOR AD=&H3000 TO &H30FF
20   POKE AD,AD MOD 256
30 NEXT
```

さきほど述べたように、書き込みのときはテキストRAMに書き込まれますから、これでテキストRAMのアドレス3000H～30FFHにデータが書き込まれました。本当に書き込まれたかどうか見てみましょう。モニタに入り、OARを30Hに設定します。(BASICで設定すると、コマンド待ちのルーチンで再びOARを書き換えてしまいます。)

リスト2-2

```
mon

h〕o70,30
h〕
```

それでは、ウィンドウを通してのぞいてみましょう。

リスト2-3

```
h〕d8000,80ff
8000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
8010 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
8020 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F    !"#$%&'()*+,-./
8030 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
8040 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
8050 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[¥]^_
8060 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F   `abcdefghijklmno
8070 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{¦}~
8080 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F   ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼
8090 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F   ┴┬┤├▔─│▕┌┐└┘╭╮╰╯
80A0 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF    。「」、・ヲァィゥェォャュョッ
80B0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF   ーアイウエオカキクケコサシスセソ
80C0 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF   タチツテトナニヌネノハヒフヘホマ
80D0 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF   ミムメモヤユヨラリルレロワン゛゜
80E0 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF   ═╞╪╡◢◣◥◤♠♥♦♣●○╱╲
80F0 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF   ╳円年月日時分秒
h〕
```

いままで述べてきたように、8000H～80FFHまでのアドレスに3000H～30FFHまでのRAMの内容が表示されています。

このウインドウ機能は書き込みのときにも働きますから、モニタの状態のまま、S, E, Fコマンド等で8000H～83FFHまでのデータを書き換えると、テキストRAMのアドレス3000H～33FFHに書き込まれることになります。8000H～83FFHのRAMに書き込まれるのではありません。

8000H～83FFHのRAMを表示／変更するときには、OARに80Hを入れ、ウインドウに映してから行ないます。

ハードウェア的には、CPUが8000H～83FFHまでの1Kバイトの間のアドレスをリード／ライトすると、

①アドレスのMSBをマスクする。

②マスクされたアドレスとOARの内容を加える。

という動作を行ない、得られたアドレスに対してCPUはアクセスします。

1 0 0 0	0 0 × ×	× × × ×	× × × ×

8000H～83FFH
までのアドレス

↓ ①

MSB MASK

0 0 0 0	0 0 × ×	× × × ×	× × × ×

マスクされたアドレス
は0～3FFHとなる

＋ ②　加える

O A R

↓

TEXT RAM アドレス

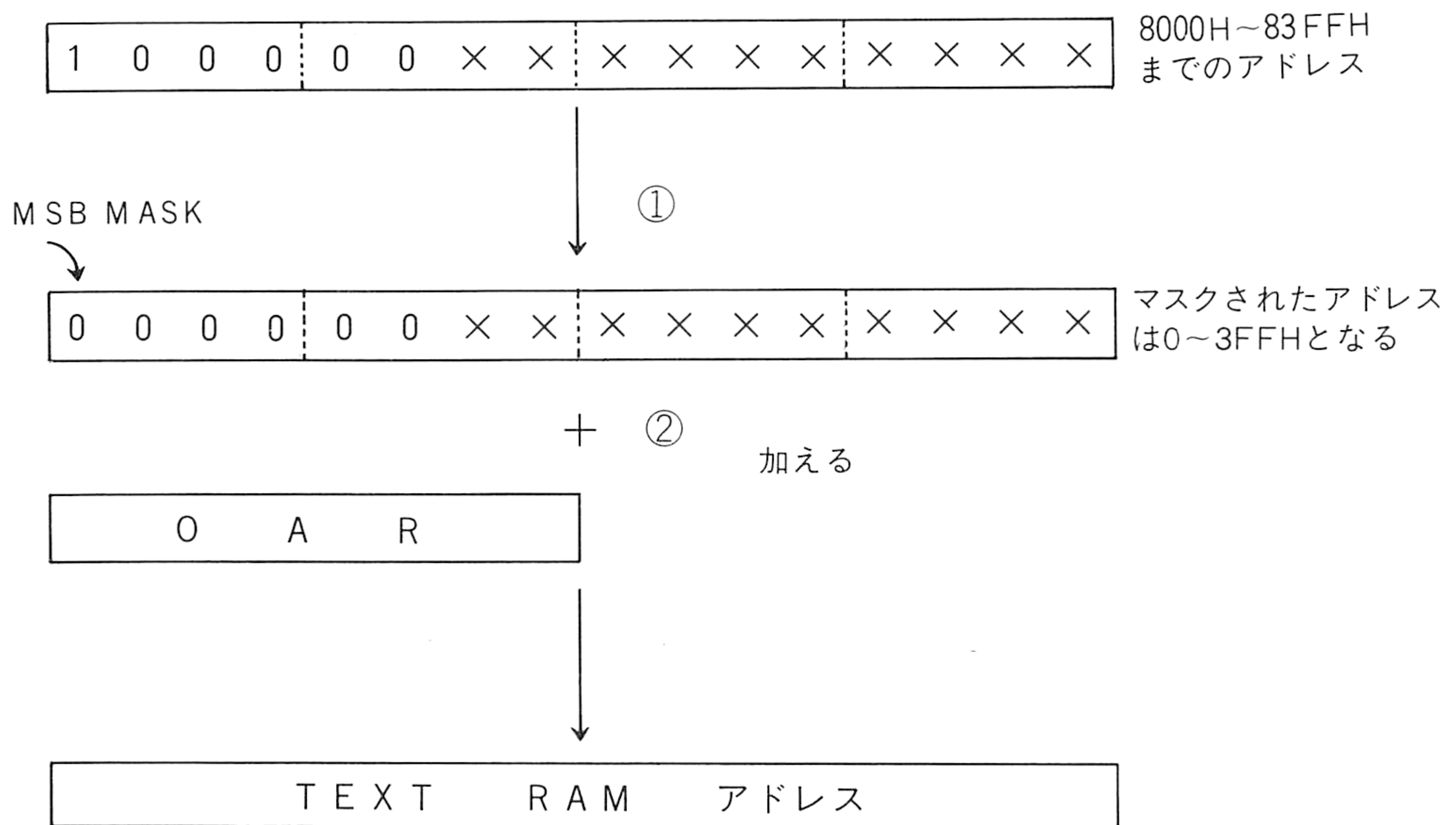



図2-4-B　ウィンドウ機能の動作

2-5　拡張ROM(4th　ROM)

「2-1メモリマップ」でも述べたようにPC-8801mkⅡは拡張ROMとして6000H～7FFFHまでの8Kバイト×7バンクを使用することができます。0000H～7FFFHまでのROMエリア32Kバイトは、バススロットの$\overline{\text{ROMKILL}}$信号をアクティブにすることによってマスクされ、バススロット上の拡張ROMがアクセス可能になります。したがって、拡張ROMボードを作製する時、OUT 71Hで拡張ROM 2～拡張ROM 8を選択するような場合、$\overline{\text{ROMKILL}}$信号がアクティブになるように設計する必要があります。

拡張ROMを選択するためのI／Oポートアドレスは次のようになっています。

ポート 71H (IN,OUT)

7	6	5	4	3	2	1	0
$\overline{\text{4th}}$ ROM 8	$\overline{\text{4th}}$ ROM 7	$\overline{\text{4th}}$ ROM 6	$\overline{\text{4th}}$ ROM 5	$\overline{\text{4th}}$ ROM 4	$\overline{\text{4th}}$ ROM 3	$\overline{\text{4th}}$ ROM 2	$\overline{\text{4th}}$ ROM 1

0：Enable　(セレクトしたいROMのビットだけを0
1：Disable　にし，残りを全て1にする。)

図2-5-A　拡張ROM　I／Oポート

たとえば、ROM 2を選択するためには、次のようにします。

```
LD      A, 0FDH
OUT     (71H), A
```

このときのメモリマップは図2-5-Bのようになります。

4th ROM 1
7FFF
6000
N88-BASIC ROM
0000
4th ROM

4th ROM 2〜8
7FFF
6000
0000
4th ROM

（点線枠）はアクセスできないことを示す。

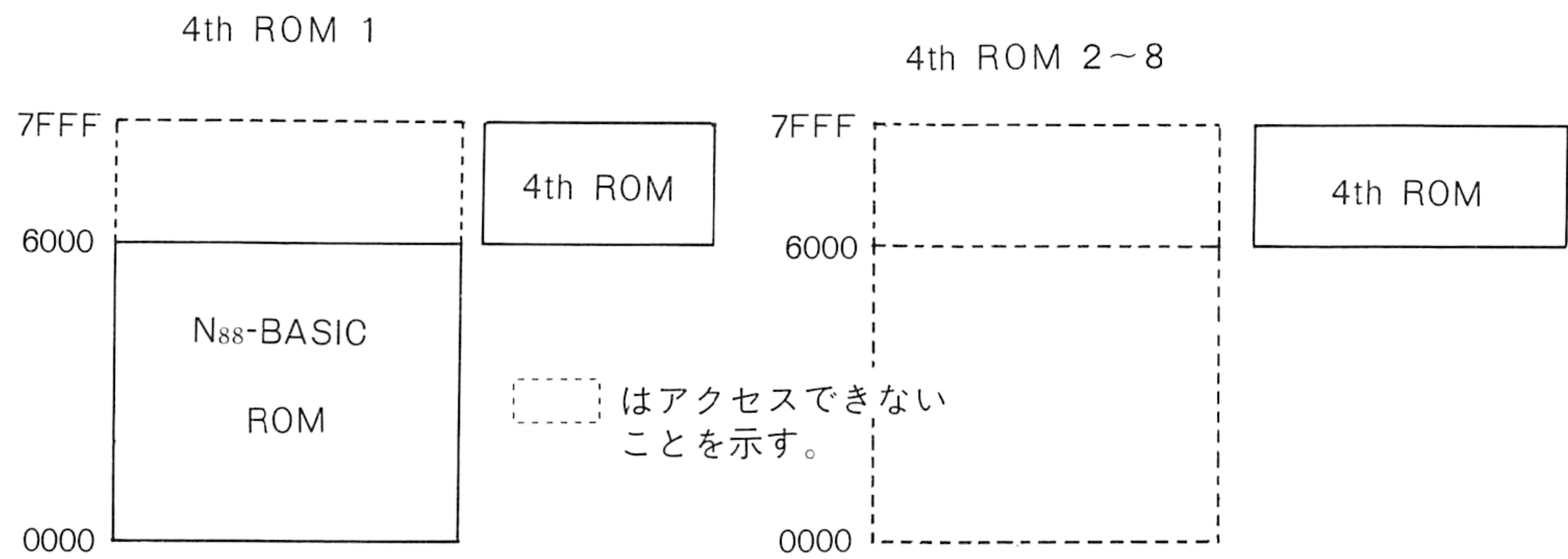



図2-5-B　拡張ROM使用時のメモリマップ

≪注意≫

①4 thROM　1というのはN88-BASICインタプリタの一部として内部に実装されています。このROMを選択した場合には内部的にROMKILL信号を用いずに処理されるため、0000H〜5FFFHのROMは禁止されません。

②4 thROMの選択をやめるときには、71HにFFHをOUTします。

また、リセット時には、OUT 71Hを行なわずに拡張ROMに制御を移すこともできます。拡張ROMの先頭、つまり6000H番地に52H(“R”)，6001H番地に34H(“4”)を書くというのがそれです。N88-BASICはリセット後のイニシャライズ時に、ROM 2からROM 8まで順に拡張ROMの6000H，6001Hをチェックし、52H，34Hが書かれていると、そのROMの6002Hにジャンプします。

拡張ROM（リセット時のイニシャライズ処理が必要な場合の使い方）

アドレス	内容	備考
6000	52H	'R'
6001	34H	'4'
6002		←ここへ制御が移る
	⋮ 処理ルーチン ⋮	
	C3H	JP 154Hとしておくともとのチェックルーチンへ戻り次のオプションROMをチェックする。
	54H	
	01H	

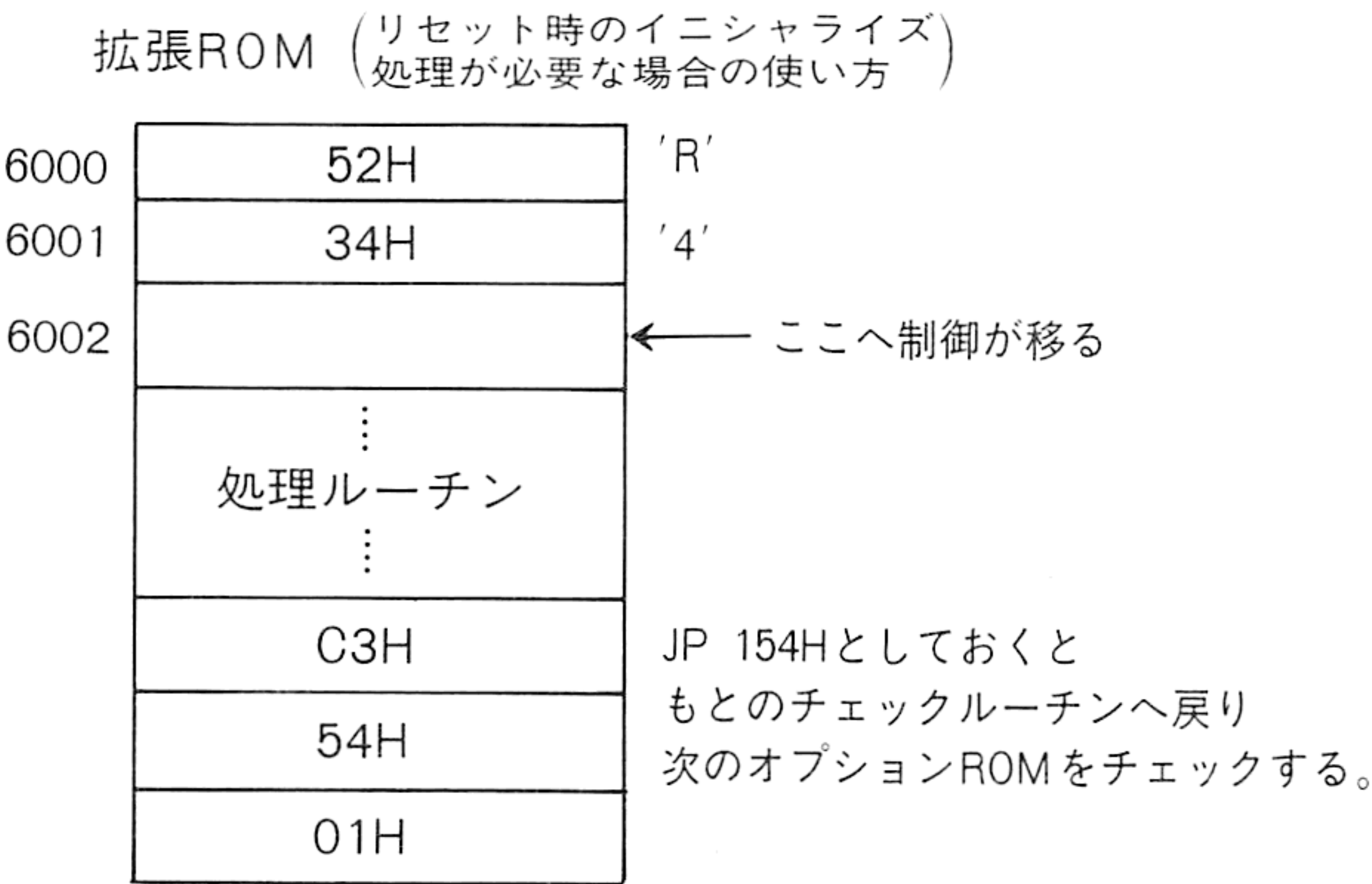
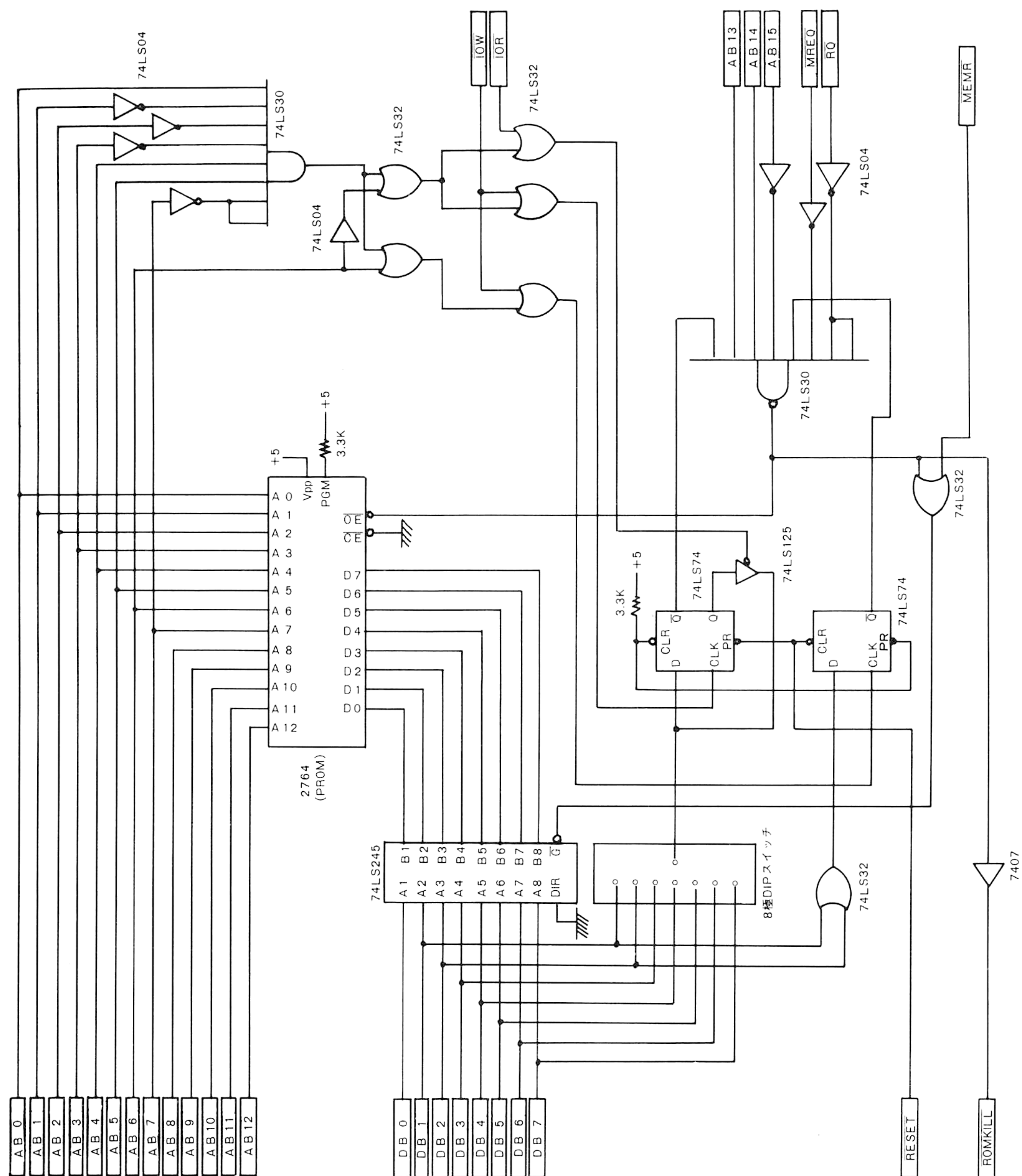



図2-5-C　拡張ROMのイニシャライズ

図2-5-Dに拡張ROMボードの回路例を上げておきます。

図２－５－D　　拡張ROMボードの回路例

2-6　拡張RAM

拡張RAMは0000H〜7FFFHの32Kバイトに割り当てられます。このときのメモリマップは右のようになります。

つまり64KバイトALL RAMになるわけで、モード2と同じようなものです。当然拡張RAMを使用するプログラムにも、モード2のときと同じ注意が必要です。市販されている拡張RAM用ボードとしては、PC-8012-02とPC-8801-02Nがあります。PC-8801-02Nは、NEC製CP/Mを使用した時にはRAMディスクとして使えますので、PC-8801mk IIにはこの方がよいでしょう。

アドレス	
FFFF〜8000	メインRAM
8000〜0000	拡張RAM

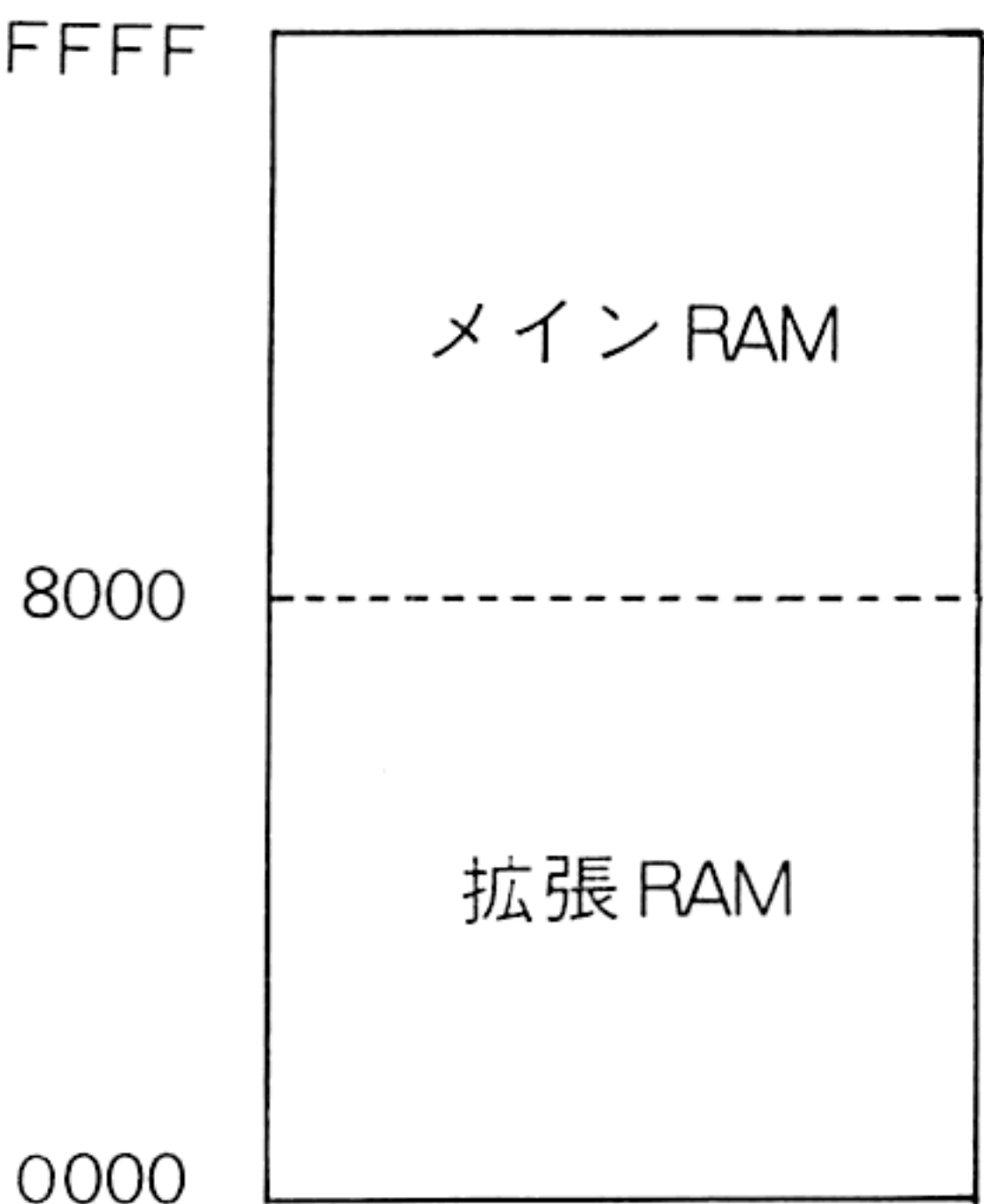



図2-6-A　拡張RAM使用時のメモリマップ

(1)　PC-8012-02

PC-8012-02は32KバイトのRAMボードです。拡張RAMのエリアは0000H〜7FFFHの32Kバイトですから、ちょうどの大きさです。PC-8012-02はバンクナンバー設定のためのジャンパがついていて、4つのバンクに設定できるので、最大4枚までを使用できるのですが、PC-8801mkIIはスロットが3つしかありませんので、3枚までしか使えません。

PC-8012-02のバンク制御はポートE2Hを用います。

ポート E2H (OUT)

書き込み制御				読み出し制御			
7	6	5	4	3	2	1	0
バンク3	バンク2	バンク1	バンク0	バンク3	バンク2	バンク1	バンク0

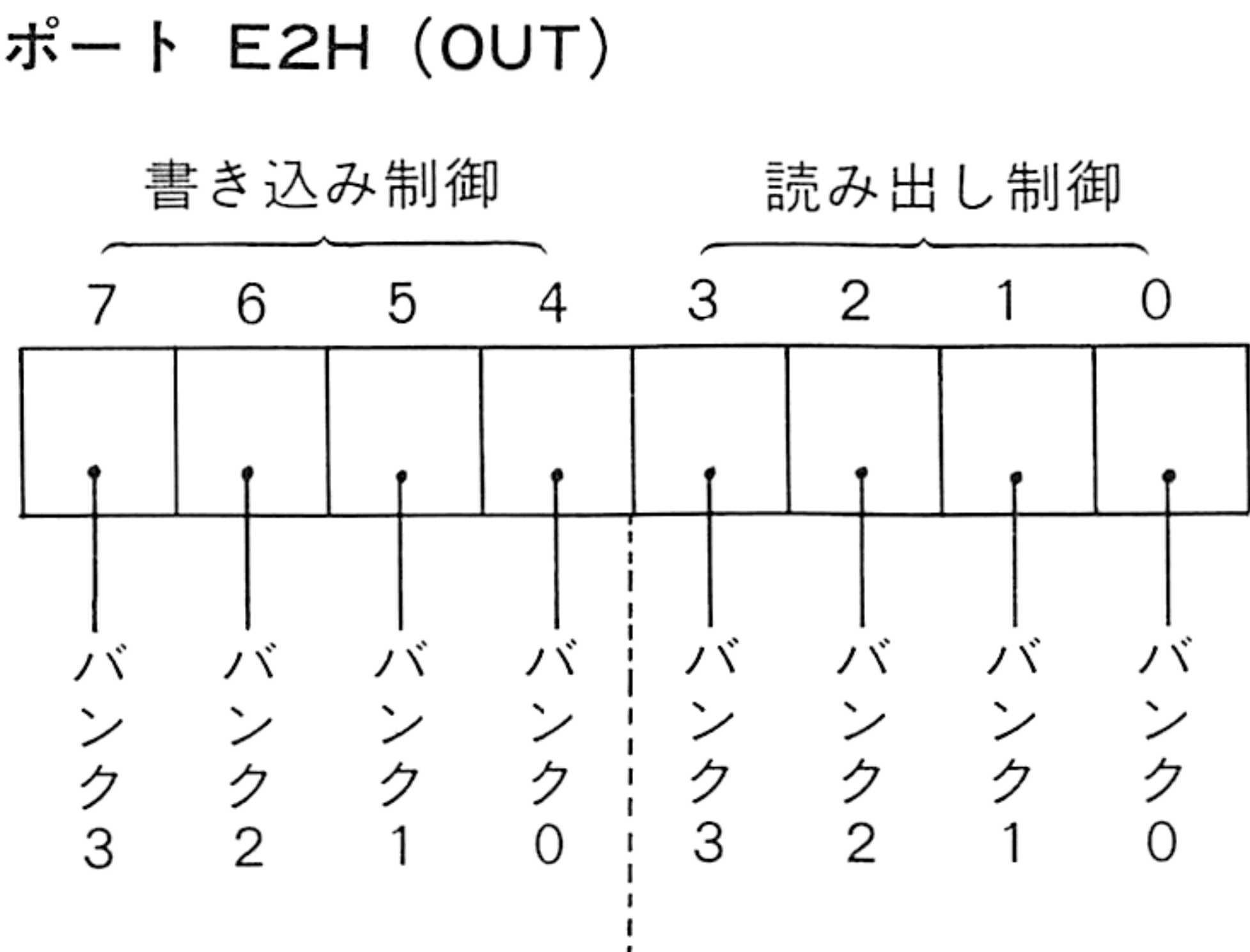



図2-6-B　PC-8012-02のバンク制御ポート

≪注意≫1度に2つ以上のバンクを読み出し許可にすることはできません。

たとえばバンク0をリード/ライト許可するときには、

```
LD    A, 11H
OUT   (0E2H), A
```

とします。こうするとバンク1に設定されているPC-8012-02のリード/ライトが許可され、同時にPC-8801mkIIの0000H〜7FFFHのROM, RAMはアクセスできなくなります。また、PC-8012-02はリードとライトを別々に制御できるようになっていますが、PC-8801mkIIで

は、テキストRAMがあるために、書き込みだけを許可することはできません。PC-8001(mkⅡ)に使用した場合には書き込みだけを許可することができるため、BASICのPOKE文で拡張RAMにデータを書き込むことができましたが、PC-8801mkⅡではこういうことはできません。

また、現在どのバンクの読み出しが許可されているかを知ることもできます。これもポートE2Hを使います。

ポート E2H (IN)

7	6	5	4	3	2	1	0
1	1	1	1	バンク3	バンク2	バンク1	バンク0

1：読み出し禁止状態
0：読み出し許可状態
(OUTの時と逆です)

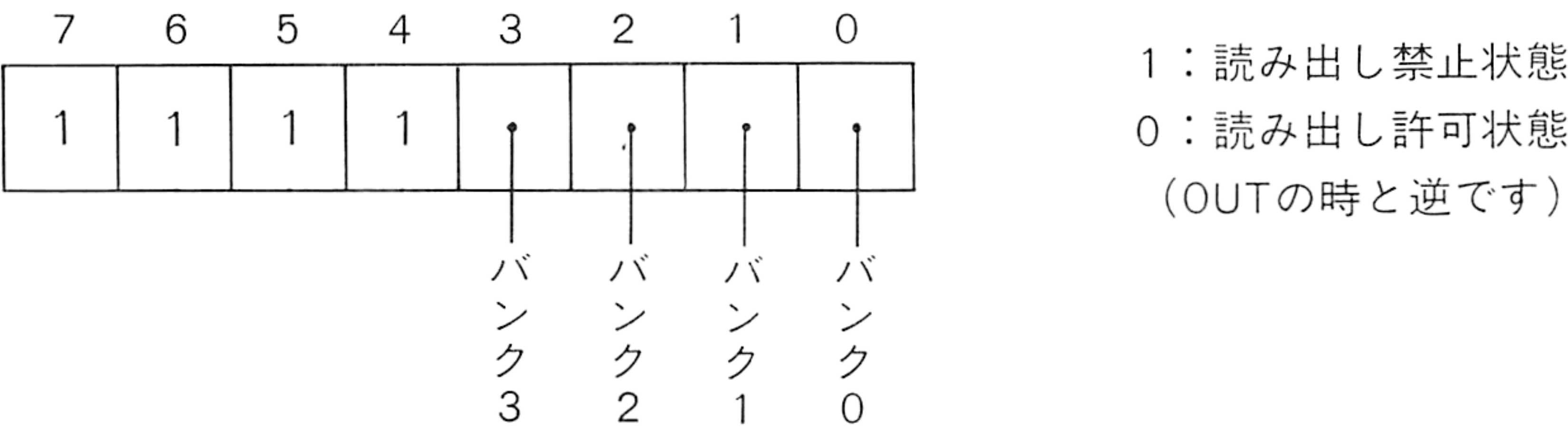



図2-6-C PC-8012-02のバンクステータス

(2) PC-8801-02N

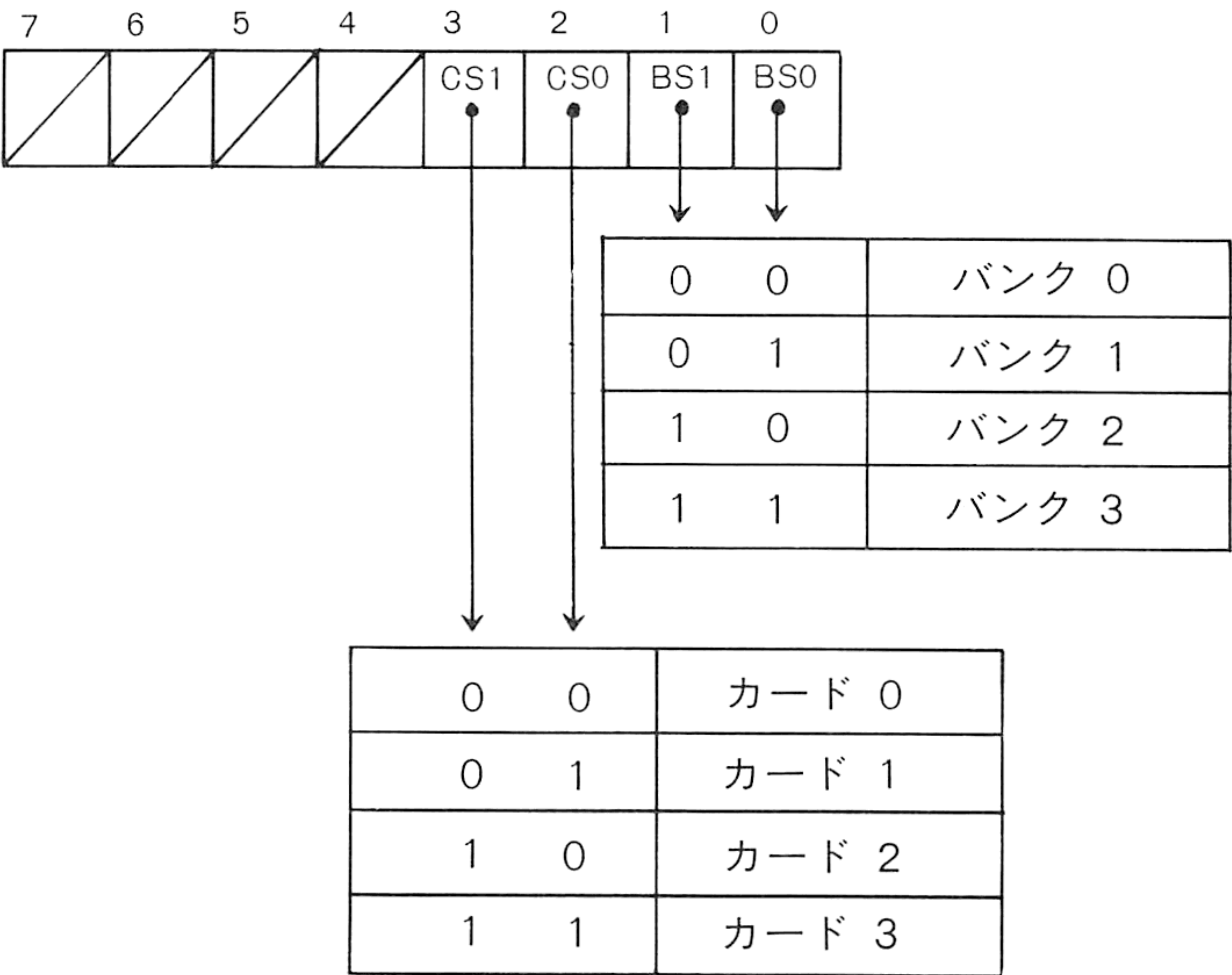
PC-8801-02Nは128Kバイトのメモリを持っていて、32Kバイトずつ、4バンクに分けて使用します。つまりPC-8012-02の4枚分ということです。また、PC-8801-02Nにはカードナンバー設定用のジャンパーコネクタがあり、最大4枚(合計512Kバイト)を使用することができます。(PC-8801mkⅡでは3枚まで)

PC-8801-02Nのバンク制御にはポートE3H, E2Hを使います。

ポート E3H (IN／OUT) カード・バンク制御ポート

7	6	5	4	3	2	1	0
/	/	/	/	CS1	CS0	BS1	BS0

BS1	BS0	
0	0	バンク 0
0	1	バンク 1
1	0	バンク 2
1	1	バンク 3

CS1	CS0	
0	0	カード 0
0	1	カード 1
1	0	カード 2
1	1	カード 3

ポート　E2H（IN／OUT）リード／ライト制御ポート

7	6	5	4	3	2	1	0
／	／	／	●	／	／	／	●

ビット0
出力：読み出し許可／禁止
入力：読み出し禁止状態／許可状態

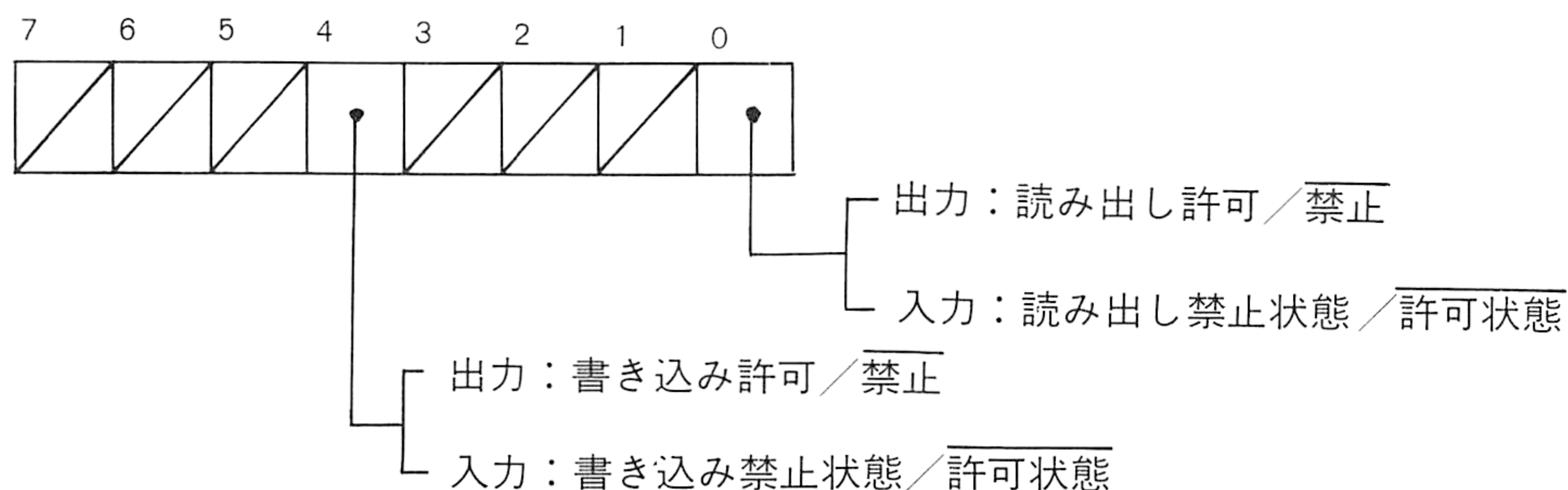
ビット4
出力：書き込み許可／禁止
入力：書き込み禁止状態／許可状態

※PC-8801mkIIでは書き込みのみ許可モードで使用してはいけません

図2-6-D　PC-8801-02Nのバンク制御

たとえば、カード0、バンク2をリード／ライト許可にするときには、

①E3Hに02HをOUTする

②E2Hに11HをOUTする

とします。またPC-8801-02NはPC-8012-02のように複数のバンクを同時に書き込み許可の状態にする機能はありません。

現在セレクトされているバンクのリード／ライト状態はポートE2HをINすることにより知ることができます。このとき、OUTのときとはビットの値が逆だということに注意しなければなりません。

以上のことをPC-8012-02と比べてみると、ポートE3Hに00をOUTするとPC-8801-02Nのカード0、バンク0はPC-8012-02のバンク0と同じように扱えることがわかります。

2-7 すべてのメモリのPEEK／POKEプログラム（拡張PEEK／POKE）

今まで述べてきた方法で、グラフィック画面以外のすべてのメモリが読み書きできますが、これをBASICで行なうことはできません。そこで、これをBASICで手軽に行なえるようにする機械語のプログラムを紹介しましょう。このプログラムを用いるとPC-8801mkⅡに標準装備されているすべてのメモリを読み書きできるようになります。なお、グラフィックVRAMと画面との関係は第7章を見てください。

（1）拡張命令（CMDシリーズ）を使用しないとき

①まず、Clear，&HE3FF[↵]を行なってからリスト2-4の機械語プログラムをモニタで正確に入力してください。入力し終わったら必ずセーブしてください。テープのときはモニタのWコマンド、ディスクのときはBSAVE文で行なうのがよいでしょう。

②モニタで、「GE400[↵]」と行ない、[CTRL]+[B]でBASICに戻ります。

③これで準備できました。使用方法はPEEK／POKEを拡張した形で行ないます。

<PEEKのとき>

PEEK（アドレス，"バンク名"）

<POKEのとき>

POKE　アドレス，データ，"バンク名"

アドレスとデータは今までと同じです。バンク名には6つあります。バンク名とそのときのメモリマップを示します。

	" "（スペース）バンク名または省略	"N"	"T"	"0"	"1"	"2"	
FFFF	メインRAM	メインRAM	メインRAM	ブルー GVRAM0	レッド GVRAM1	グリーン GVRAM2	FFFF
				メインRAM	メインRAM	メインRAM	
	ウィンドウ						
8000	N88-BASIC ROM	N-BASIC ROM	テキストRAM	4th ROM	4th ROM	4th ROM	8000
				N88-BASIC	N88-BASIC	N88-BASIC	6000
0000							0000

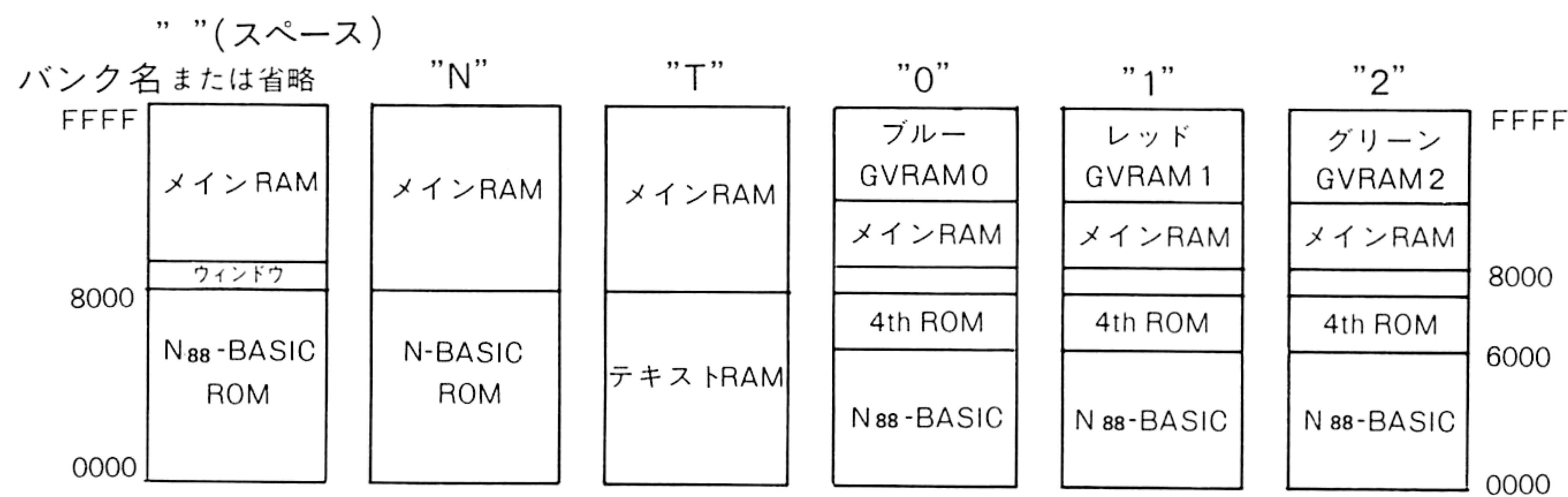



図2-7-A　バンク名とメモリマップ

たとえば、テキストRAMの123H番地を読むときには、？PEEK（&H123，"T"）[↵]と行ないます。また、POKE　&HC000，255，"2"[↵]とすると、グラフィックVRAMのグリーンバンクのアドレスC000Hにデータ&HFFが書き込まれ、画面の左上隅に横棒が現われます。また、通常のPEEK文、POKE文はこれまでどおり行なえます。

リスト　2-4

```
E400 3A 74 E5 B7 20 34 F3 21 27 ED 11 74 E5 01 03 00
E410 ED B0 21 9F ED 11 77 E5 01 03 00 ED B0 3E C3 32
E420 27 ED 32 9F ED 3A 27 ED 21 3C E4 22 28 ED 21 EB
E430 E4 22 A0 ED 3E 6C 32 27 E8 FB FF C9 3C 28 04 3D
E440 C3 74 E5 23 7E FE 97 28 04 2B 3E FF C9 F1 CD 77
E450 E5 3E 05 F5 D7 CF 28 CD 93 1B D5 2B D7 FE 29 28
E460 09 CF 2C CD BC E4 D1 C1 F5 D5 CF 29 D1 ED 53 71
E470 E5 F1 32 73 E5 E5 3E 02 32 BD EA 2A 71 E5 CD 28
E480 E5 CD 99 E4 4E D3 5F 3E FF D3 71 3A C2 E6 D3 31
E490 FB 69 26 00 22 41 EC E1 C9 F3 3A 73 E5 D6 03 30
E4A0 05 3E FE D3 71 C9 20 08 3A C2 E6 F6 02 D3 31 C9
E4B0 3D C0 3A C2 E6 F6 04 E6 FD D3 31 C9 CD D3 11 E5
E4C0 CD C9 56 7E B7 CA 06 0B 23 5E 23 56 1A 0E 05 FE
E4D0 20 28 15 0D FE 4E 28 10 0D FE 54 28 0B D6 30 DA
E4E0 06 0B FE 03 D2 06 0B 4F 79 E1 C9 F1 CD 77 E5 CF
E4F0 2C 3E 05 F5 CD A3 18 F5 2B D7 28 09 CF 2C CD BC

E500 E4 C1 D1 F5 C5 C1 F1 32 73 E5 D1 ED 53 71 E5 E5
E510 2A 71 E5 CD 28 E5 CD 99 E4 70 D3 5F 3E FF D3 71
E520 3A C2 E6 D3 31 FB E1 C9 3A 73 E5 FE 03 D0 11 00
E530 C0 E7 D8 F3 78 21 50 84 11 7A E5 01 06 00 ED B0
E540 E1 F5 3E D3 11 50 84 12 3A 73 E5 C6 5C 13 12 23
E550 23 23 13 01 03 00 ED B0 3E C9 12 C1 E5 2A 71 E5
E560 CD 50 84 C5 21 7A E5 11 50 84 01 06 00 ED B0 C1
E570 C9 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(2)拡張命令を同時に使用したいとき

(1)とやり方は同じなのですが、アドレスが異なります。

①clear, &HDA00↵で領域を確保し、リスト2-5のダンプリストを打ち込みます。

②初期化はh] gda00↵で行ないます。文法その他は(1)と同じです。

リスト　2-5

```
DA00 3A 74 DB B7 20 34 F3 21 27 ED 11 74 DB 01 03 00
DA10 ED B0 21 9F ED 11 77 DB 01 03 00 ED B0 3E C3 32
DA20 27 ED 32 9F ED 3A 27 ED 21 3C DA 22 28 ED 21 EB
DA30 DA 22 A0 ED 3E 6C 32 27 E8 FB FF C9 3C 28 04 3D
DA40 C3 74 DB 23 7E FE 97 28 04 2B 3E FF C9 F1 CD 77
DA50 DB 3E 05 F5 D7 CF 28 CD 93 1B D5 2B D7 FE 29 28
DA60 09 CF 2C CD BC DA D1 C1 F5 D5 CF 29 D1 ED 53 71
DA70 DB F1 32 73 DB E5 3E 02 32 BD EA 2A 71 DB CD 28
DA80 DB CD 99 DA 4E D3 5F 3E FF D3 71 3A C2 E6 D3 31
DA90 FB 69 26 00 22 41 EC E1 C9 F3 3A 73 DB D6 03 30
DAA0 05 3E FE D3 71 C9 20 08 3A C2 E6 F6 02 D3 31 C9
DAB0 3D C0 3A C2 E6 F6 04 E6 FD D3 31 C9 CD D3 11 E5
DAC0 CD C9 56 7E B7 CA 06 0B 23 5E 23 56 1A 0E 05 FE
DAD0 20 28 15 0D FE 4E 28 10 0D FE 54 28 0B D6 30 DA
DAE0 06 0B FE 03 D2 06 0B 4F 79 E1 C9 F1 CD 77 DB CF
DAF0 2C 3E 05 F5 CD A3 18 F5 2B D7 28 09 CF 2C CD BC

DB00 DA C1 D1 F5 C5 C1 F1 32 73 DB D1 ED 53 71 DB E5
DB10 2A 71 DB CD 28 DB CD 99 DA 70 D3 5F 3E FF D3 71
DB20 3A C2 E6 D3 31 FB E1 C9 3A 73 DB FE 03 D0 11 00
DB30 C0 E7 D8 F3 78 21 50 84 11 7A DB 01 06 00 ED B0
DB40 E1 F5 3E D3 11 50 84 12 3A 73 DB C6 5C 13 12 23
DB50 23 23 13 01 03 00 ED B0 3E C9 12 C1 E5 2A 71 DB
DB60 CD 50 84 C5 21 7A DB 11 50 84 01 06 00 ED B0 C1
DB70 C9 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

第３章　N$_{88}$DISK-BASIC内部構造

3－1　N$_{88}$DISK-BASICメモリ・マップ

N$_{88}$DISK-BASICでは多くのROM，RAMのうち、以下のものを使用しています。

アドレス	メインRAM			
FFFF	メインRAM	GVRAM0	GVRAM1	GVRAM2
C000		C000	C000	C000
8000		7FFF	7FFF	7FFF
6000	テキスト RAM	N$_{88}$-BASIC ROM	モニタ	4th ROM
			N-BASIC ROM	
0000		0000		

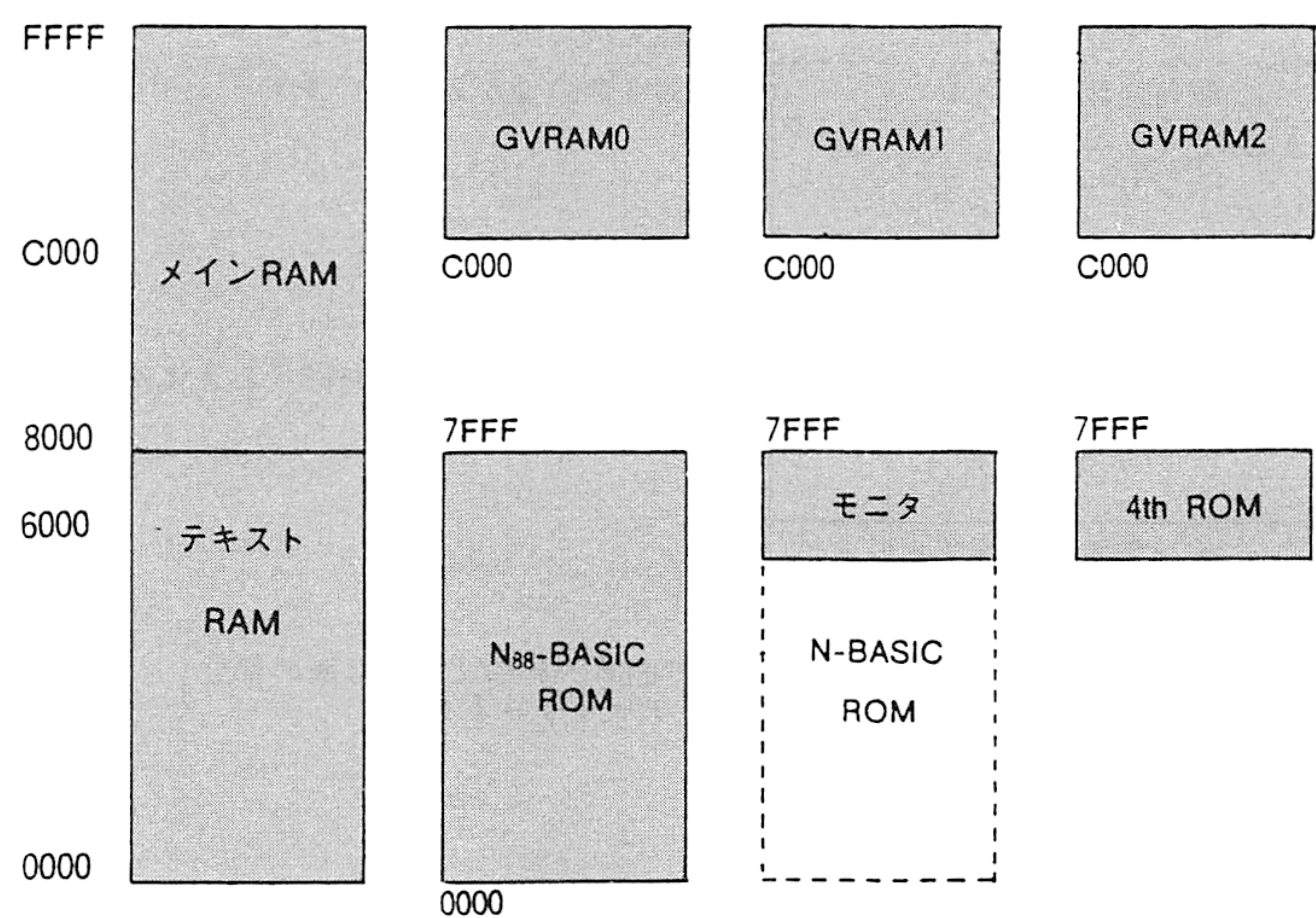



図３－１－A　N88-DISK BASICメモリマップ

3-1-1 メインRAM

メインRAMはBASICの変数領域、ワークエリア、テキストVRAMなどに使用されています。詳しいメモリマップはこうなっています。

アドレス	領域	説明
8000	テキスト・ウィンドウ	←BASICテキストの一部が現われます。
8400	★ディスクコード	←ディスク関係の処理ルーチンです。
	★ドライブ ポインタ	
	ファイル ポインタ	
	★ドライブ テーブル	
	ファイルバッファ	
	ラベルテーブル	←ラベルが登録されます。
	単純変数テーブル	←単純変数の値が入っています。
	配列変数テーブル	←配列変数の値が入っています。
	フリースペース	
	文字列エリア	←文字列が入っています。
	スタックエリア	
	ユーザー機械語エリア	←ユーザーが作成した機械語をおきます。
E600	ワークエリア	
F300	インタラプトベクトル	
F320	空きエリア	
F3C8	テキストURAM	(N88ROM-BASICでは★の部分がなくなります。)
FFF8	空きエリア	
FFFF		

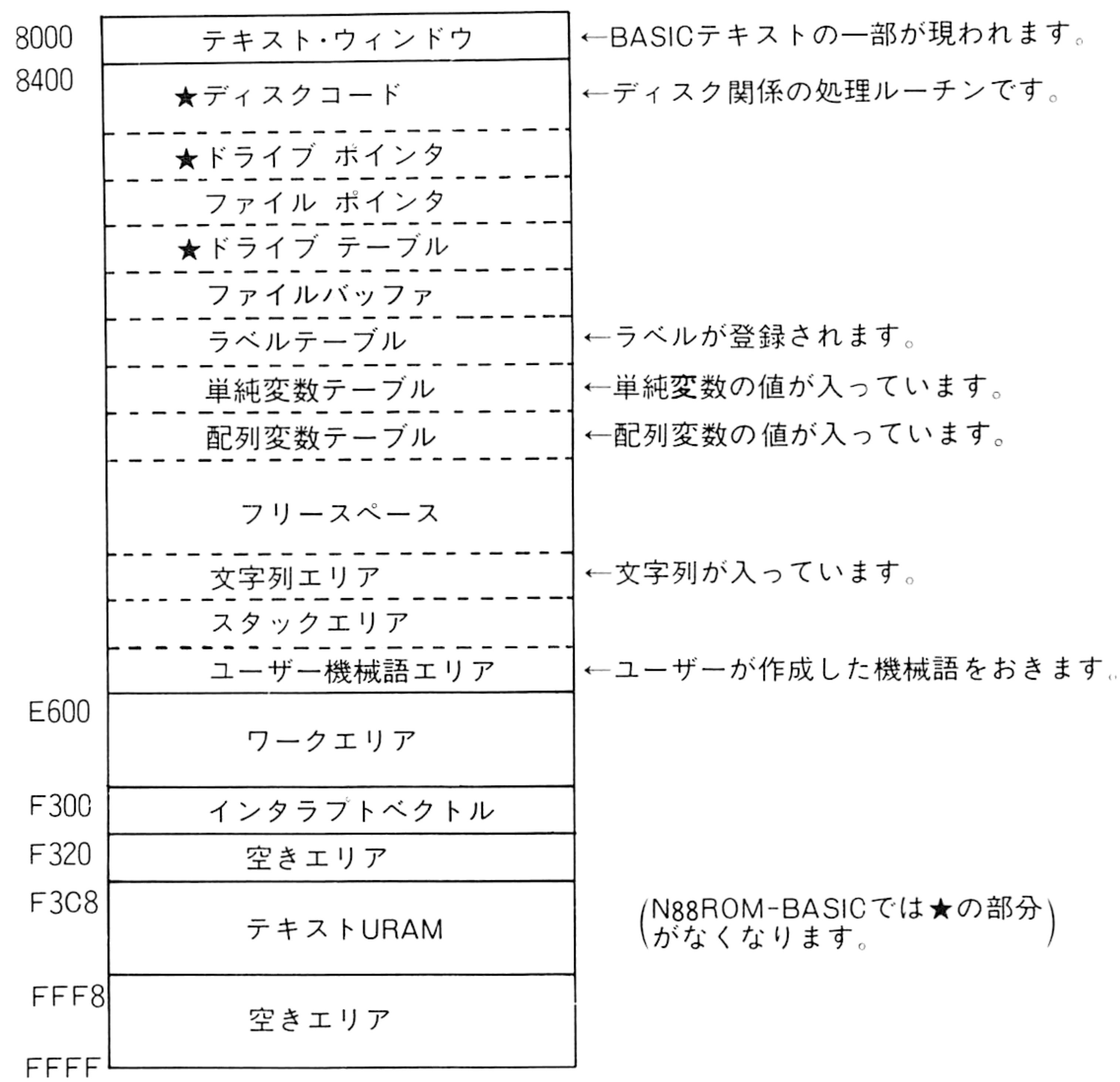



図3-1-B メインRAMメモリマップ

8400H～E5FFHの領域は、どこからどこまでが何、とはっきり決められているわけではありません。場合場合によって変化します。そのために、各々の領域の場所を示すポインタがワークエリアの中にあります。

それぞれのポインタのアドレスとその示している場所は図3-1-Cの通りです。

アドレス	領域	ポインタアドレス(16進)
8400	ディスクコード	
	ドライブポインタ	←(EC81,82)
	ファイルポインタ	←(EC7F,80)
	ドライブテーブル	
	ファイルバッファ	
	ラベルテーブル	←(EB16,17)
	↓ 単純変数テーブル	←(EB1B,1C)
	↓ 配列変数テーブル	←(EB1D,1E)
FRE(0)	↓ フリーエリア ↑	←(EB1F,20) ←(EAF1,F2)
	文字列	←(EACC,CD)
	スタックエリア	←(E654,55)
	ユーザー機械語エリア	←(E7E8,E9)
E600	ワークエリア	

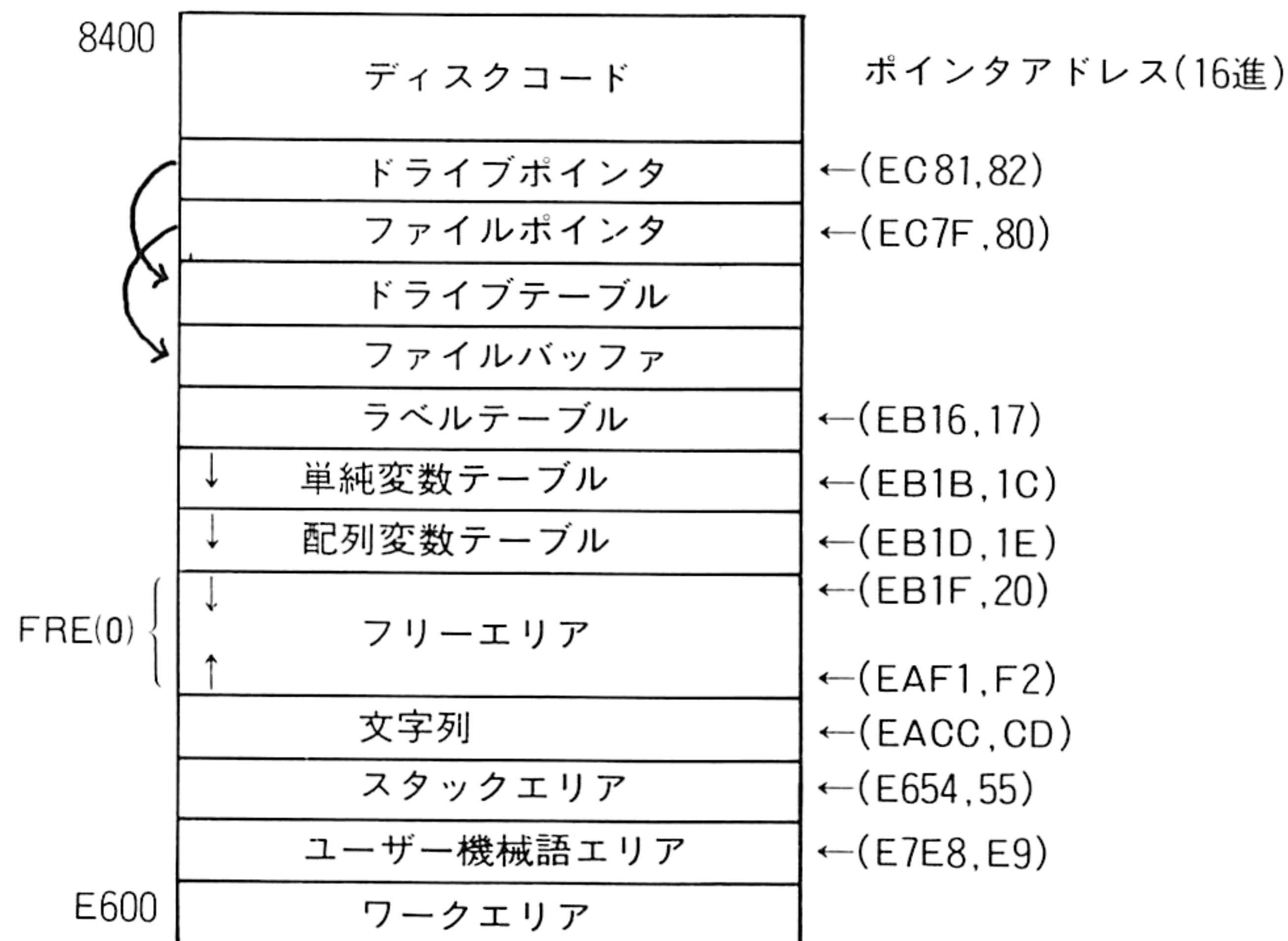



図3-1-C 領域の場所を示すポインタ

これらのポインタのうち、ドライブポインタテーブルの先頭、ファイルポインタテーブルの先頭、ラベルテーブルの先頭、使用可能エリアの最後はリセット時に決められます。また、文字列格納エリアの最後、スタックエリアの最後はCLEAR文によって変更できます。他のポインタはその時々にダイナミックに変化します。

例えば、CLEAR, &HDFFF, 1000⏎と行なうと、スタックエリアの最後はDFFFH, 文字列格納エリアの最後はDFFFH-1000＝DC17Hとなります。実際にモニタで見てみましょう。

リスト 3-1

```
clear,&hdfff,1000
Ok
mon


h] de654
E654 FF DF FF FF 01 00 1C 6F 4E 45 43 30 30 30 30 30
h] deacc
EACC 17 DC D0 EA 05 1A 03 02 36 DA 03 CD 80 00 00 00
```

3-1-2 テキストRAM

テキストRAMには通常、BASICプログラムのみが置かれてます。プログラムの後は空きエリアです。

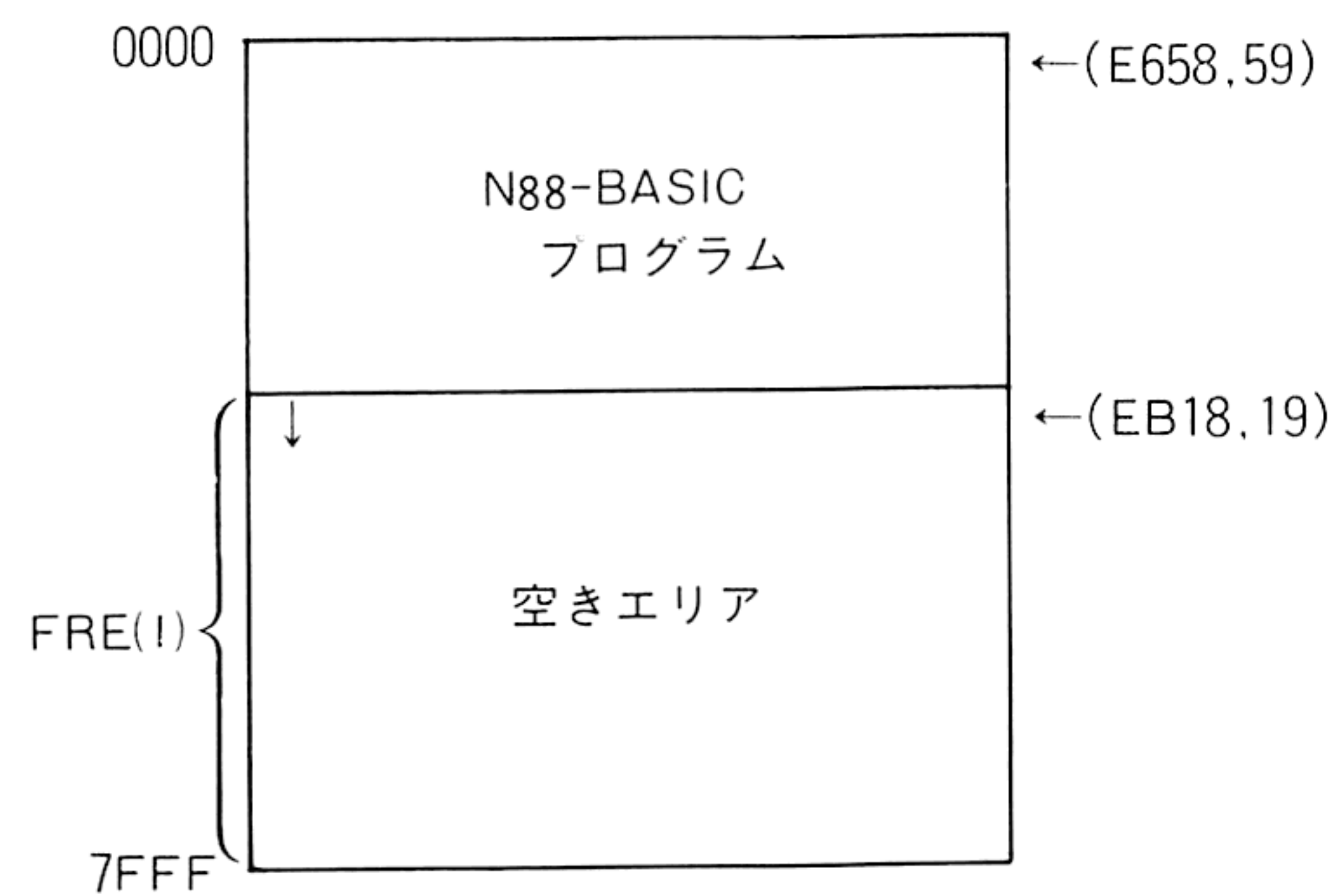



図3-1-D テキストRAMメモリマップ

3-1-3 メモリマップの変化

(1)ファイルポインタとファイルバッファ

ファイルポインタとファイルバッファは、電源ON時の「How many files？」に対する答えによって変わってきます。入力した数値をNとすると、＃0～＃NのN＋1個のポインタとバッファがとられます。ただ⏎キーを押した時は、ROM BASICのとき2個(1を答えたのと同じ)、DISK-BASICのときディスクドライブの数＋1個のポインタとバッファがとられます。

この数はEC7EH番地に入っています。EC7EH番地が3のときは＃0～＃3の4個のポインタとバッファがあるということです。ただし、＃0のバッファは普通の用途には使えず、DSKI $, DSKO$などでのみ使うことができます。つまり、FIELD＃0は可能ですが、

OPEN～AS＃0はできません。

下図はオープンできるファイル数が1個の場合と3個の場合のメモリ・マップを示したものです。

file数＝1

8000
テキスト・ウィンドウ
ディスクコード
ドライブポインタ
＃0ポインタ
＃1ポインタ
ドライブ
テーブル
＃0バッファ
＃1バッファ
ラベルテーブル

file数＝3

テキスト・ウィンドウ
ディスクコード
ドライブポインタ
＃0ポインタ
＃1ポインタ
＃2ポインタ
＃3ポインタ
ドライブ
テーブル
＃0バッファ
＃1バッファ
＃2バッファ
＃3バッファ
ラベルテーブル

ポインタは各々2バイト

各バッファは9バイトのファイルコントロールブロックと256バイトのバッファとから構成されます。
詳細は第4章を参照して下さい。

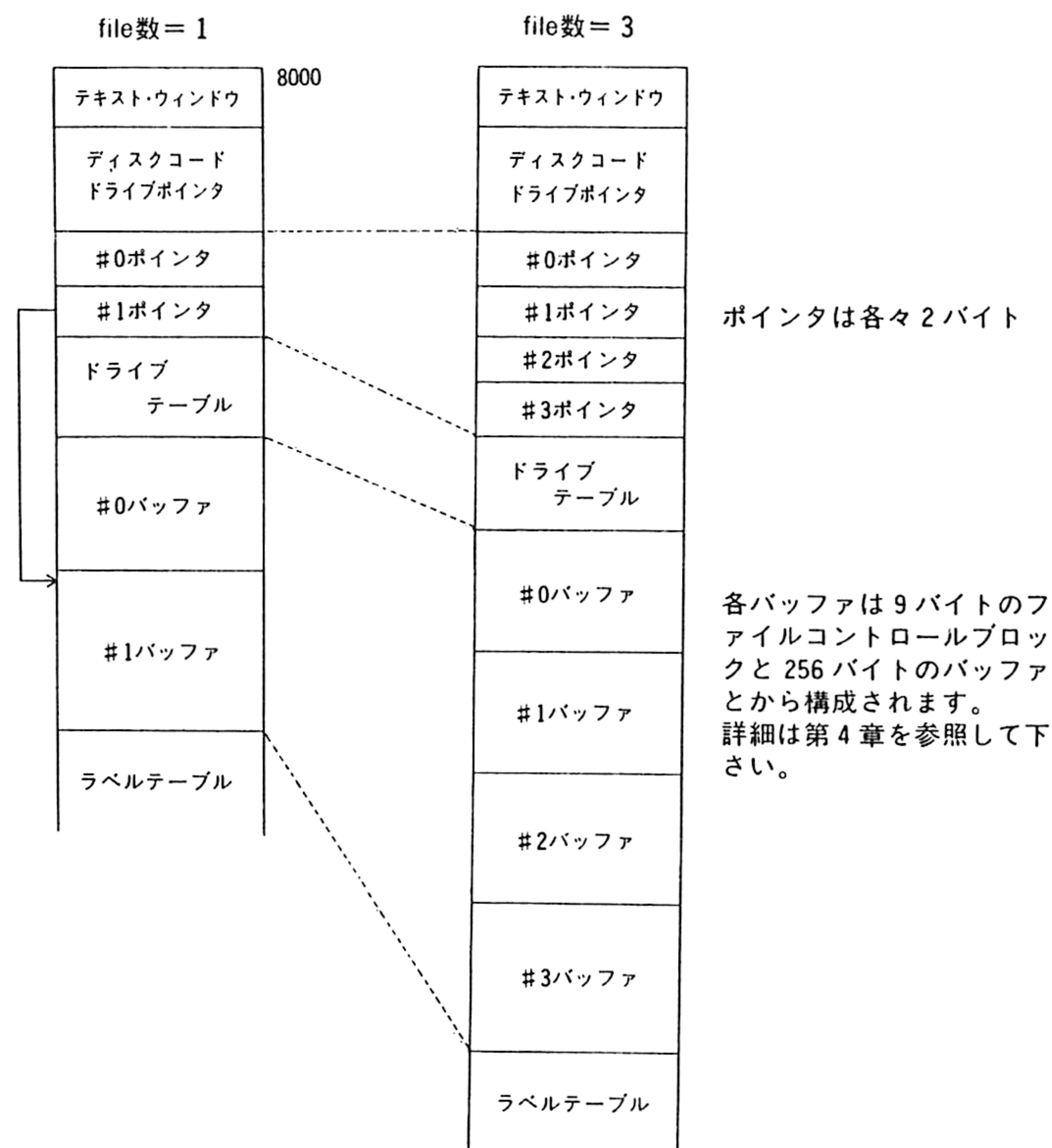



図3-1-E　ファイルポインタとファイルバッファ

(2)ラベル・変数テーブル

ラベル・変数テーブルは、プログラムによって、またその実行中に、刻々と変化します。

ラベルテーブルは、RUNの最初、CLEAR，LOAD後などの時にプログラム中からラベルを集めて、一気に作られます。この時、この後の変数テーブルは消されてしまいます。またRUNの時は最初にテーブルがつくられてしまうわけですから、実行中にラベルテーブルが変化することはありません。

これに対して変数テーブルは、プログラム実行中に新しい変数が現われるたびに変化します。その概念図を図3-1-Fに示します。

8000←　→FFFF

プログラム実行前　ファイルバッファ　フリースペース
RUN直後　〃　ラベルテーブル　フリースペース
変数をいくつか使った　〃　〃　単純変数　配列変数　フリースペース
新しい変数を使った　〃　〃　単純変数　配列変数　フリースペース
配列をERASEした　〃　〃　単純変数　配列変数　フリースペース

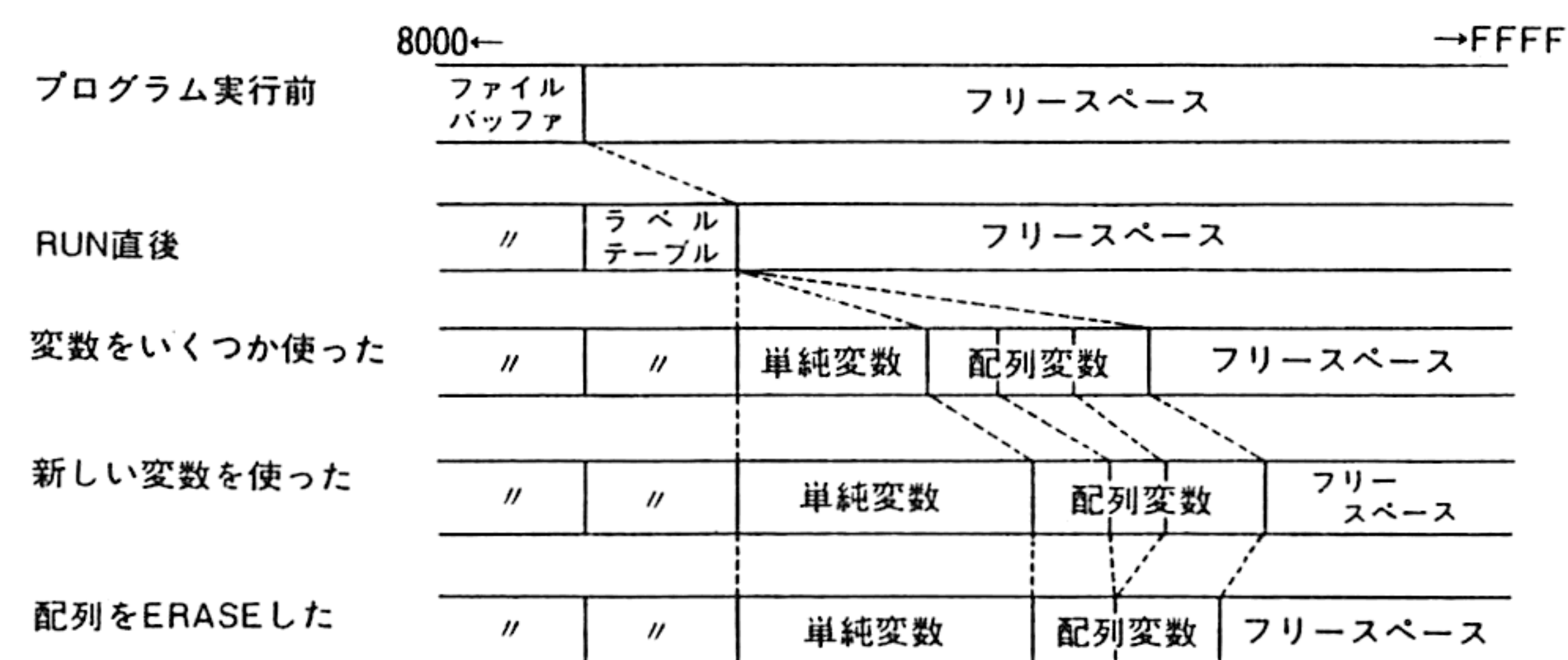



図3-1-F　ラベル・変数テーブル

(3)文字列領域

図3-1-Gのように、文字列はメモリの後のアドレスから順に格納されていきます。このとき、同じ文字変数に新しい文字列を代入すると、古い文字列はそのままで、新たに文字列領域を増やして、そこに新しい文字列を書き込みます。そのため、文字列領域内には、もう使われていない、ゴミとなった文字列がたまっていきます。

これはどうするのかというと、いずれ文字列領域が増えてフリースペースがなくなった時、新しい文字列を作ろうとしても作る場所がありませんから、その時にこれらのゴミを処分します。ゴミの部分を消して、使用中の文字列を後から順番につめていくわけです。こうして新しい文字列のためのスペースを作ります。これをガベージコレクション(Garbage collection)と呼びます。

多くの文字列を使ったプログラムを実行していると、しばらく実行が止まってしまうことがありますが、それはこのガベージコレクションを行なっているためです。また、DIM文で配列を宣言した時にフリースペースが足りない時や、PAINT文でフリースペース中にとる作業領域が足りない時もガベージコレクションを行なって、必要なメモリを確保しようとします。

なお、BASICリファレンスマニュアルにもあるとおり、FRE関数でもガベージコレクションを行ないます。これは逆にFRE関数を用いて強制的にガベージコレクションを行なわせることができるわけで、要所要所にダミーのFRE関数を入れておくことより、起こっては困るときにガベージコレクションが起こるのを防ぐことができます。

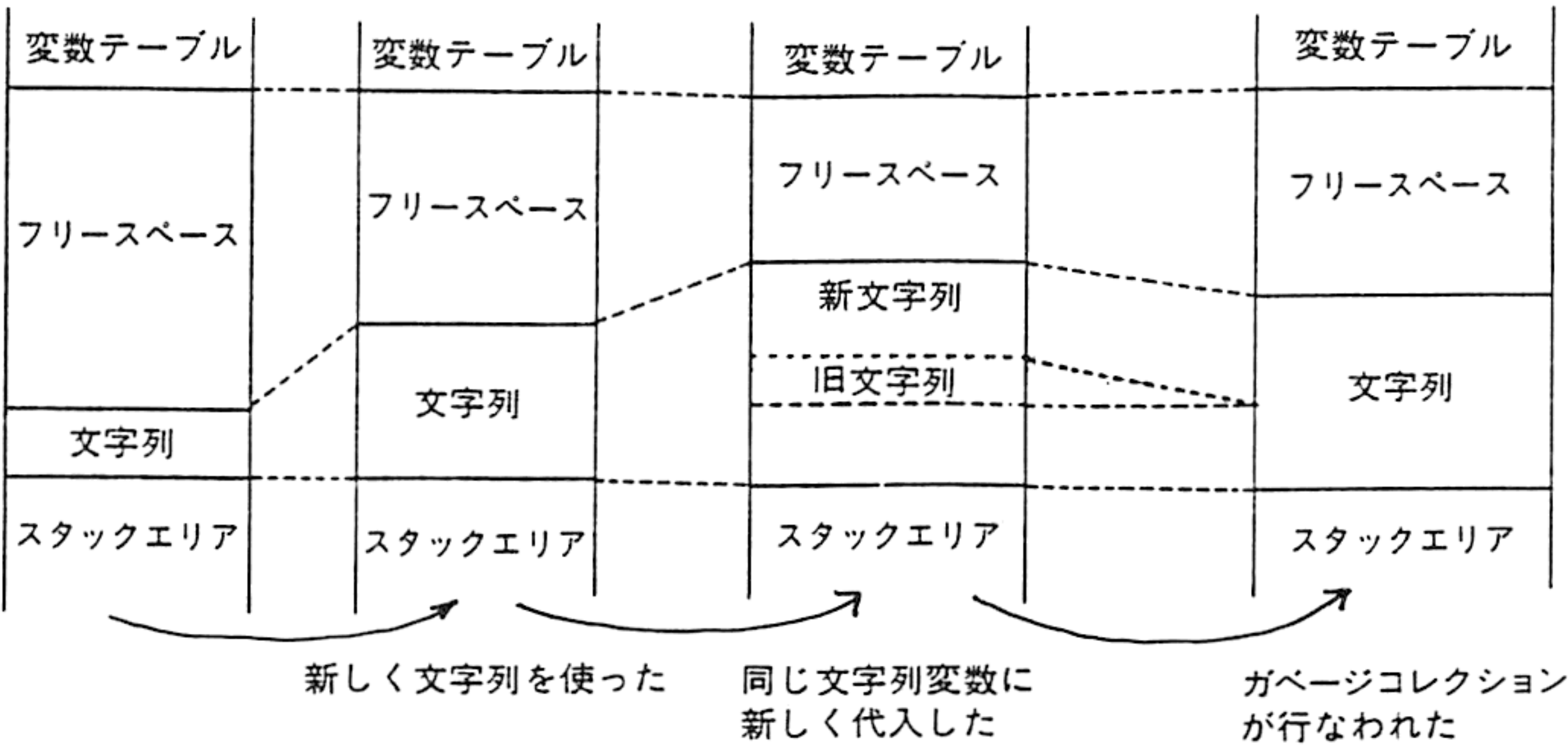



図3-1-G　文字列領域の変化

3-2　プログラムの格納状態

BASICプログラムは、テキストRAMの先頭から、行番号順に格納されています。それぞれの行はこのような形式になっています。

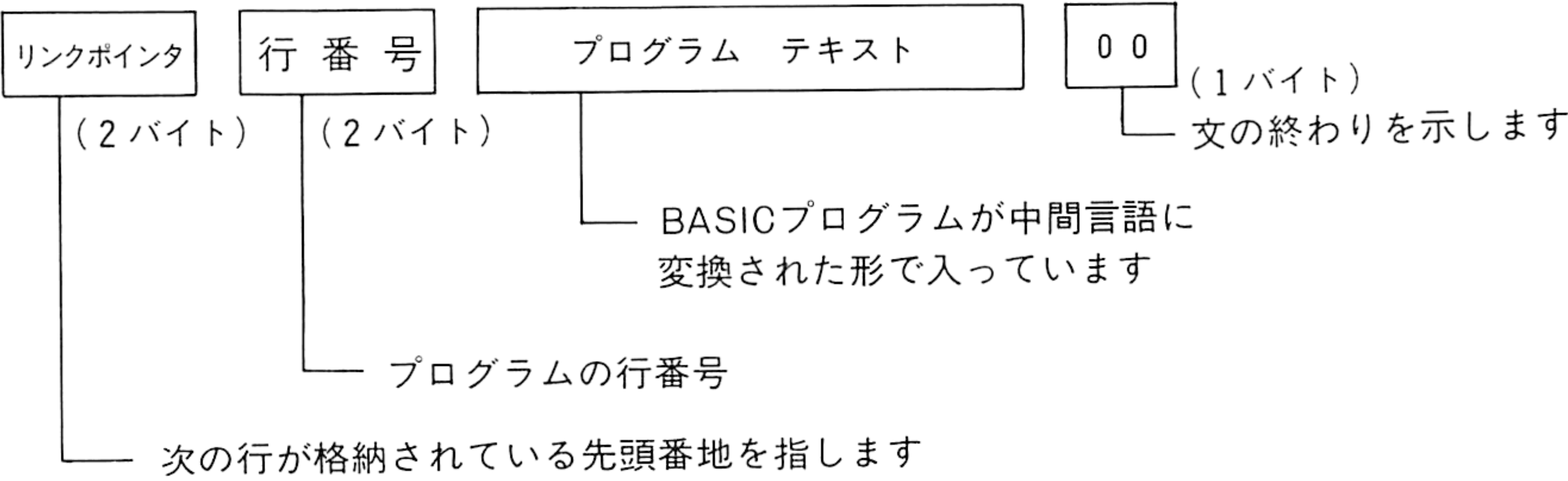



図3-2-A　行の格納形式

プログラムの終わりでは、リンクポインタの値が00となって、後に続く文がないことを示します。

次に、プログラムテキストがどのように格納されるか見てみましょう。

例として、次のプログラムを使います。

リスト　3-2

```
100 FOR I=0 TO 20 STEP .5
110  PRINT I,SQR(I)
120 NEXT I
```

テキストRAMはBASIC ROMと同じ番地にあり､直接見ることができません。そこで､テキストウィンドウを使ってのぞいてみます。

リスト　3-3

```
h] o70,0
h] d8000,8030
8000 00 18 00 64 00 82 20 49 F1 11 20 DC 20 0F 14 20
8010 DF 20 1D 00 00 00 80 00 27 00 6E 00 20 91 20 49
8020 2C FF 87 28 49 29 00 2F 00 78 00 83 20 49 00 00
8030 00
h]
```

*h] o70, 0とあるのは、0000H番地から3FFH番地をテキスト・ウィンドウに映すために、ポート70Hに、データ0をOUTする命令です。

各データは、図3-2-Bのような意味を持ちます。リストの1文字を1バイトと数えたときよりもかなり短くなっています。

```
0000 00 ※必ず00でなければならない
0001 18 ┐リンクポインタ→0018
0002 00 ┘
0003 64 ┐行番号(=100)
0004 00 ┘
0005 82 ]FOR
0006 20 ]スペース
0007 49 ] I
0008 F1 ] =
0009 11 ] 0
000A 20 ]スペース
000B DC ]TO
000C 20 ]スペース
000D 0F ┐20
000E 14 ┘
000F 20 ]スペース
0010 DF ]STEP
0011 20 ]スペース
0012 1D ┐
0013 00 │
0014 00 │0.5
0015 00 │
0016 80 ┘
0017 00 ]行の終り

0018 27 ┐リンクホインタ→0027
0019 00 ┘
001A 6E ┐行番号(=100)
001B 00 ┘
001C 20 ]スペース
001D 91 ]PRINT
001E 20 ]スペース
001F 49 ] I
0020 2C ] ,
0021 FF ┐SQR
0022 87 ┘
0023 28 ] (
0024 49 ] I
0025 29 ] )
0026 00 ]行の終り

0027 2F ┐リンクホインタ→002F
0028 00 ┘
0029 78 ┐行番号(=120)
002A 00 ┘
002B 83 ]NEXT
002C 20 ]スペース
002D 49 ] I
002E 00 ]行の終わり

002F 00 ┐プログラムの終わり
0030 00 ┘
```

図3-2-B　サンプルプログラムの格納形式

3-3　中間言語

前の節で、BASICプログラムのテキストが短縮された形で格納されていることがわかりました。これはBASICのキーワード(FOR, PRINTなど)を1バイトか2バイトの中間言語で表わしているからです。

それでは、N88-BASICでどのような中間言語が使われているのかを見てみましょう。

3-3-1　中間言語コード(00H～7FH)

中間言語コードの00Hから7FHは、数値や行番号、変数名などに使われます。

中間言語	意　　味	備　　考
OA　LF	(Line Feed) CTRL＋Jで入力	
OB　&O	続く2バイトは8進数	OB 9C 02=&01234
OC　&H	続く2バイトは16進数	OC 34 12=&H1234
OD　アドレス	続く2バイトは飛び先絶対アドレス	GOTO, GOSUB, THEN, ELSE, RESTORE などの後に続きます
OE　行番号	続く2バイトは飛び先行番号	
OF　整数（1バイト）	続く1バイトは、10〜255の整数	OF 50=80
11〜1A　整数（1ケタ）	1桁の整数（11→0.12→1.1A→9）	
1B	整数　10	通常使われていません
1C　整数（2バイト）	続く2バイトは、整数	1C D2 04=1234
1D　単精度（4バイト）	続く4バイトは、単精度定数	1D EB C0 1D 81 =1.2345
1F　倍精度（8バイト）	続く8バイトは、倍数精度定数	
20　文字	キャラクターコードに対応する文字(変数名やラベル名など)	アルファベットの小文字に変換されるため使われません。
7F		(*) 通常使われていません

3-3-2　中間言語コード(80H〜FFH)

中間言語コードの80HからFFHは、1バイトまたは2バイトで、N88-BASICのキーワードを示します。(リスト3-4, 5)

N-BASICでも似たようなキーワードと中間言語コードを持っていますが、N88-BASICと比べると対応していなかったり、バイト数が違っているものがあります。N-BASICのプログラムを「LOAD" CAS2：ファイル名"」でロードして、リストをとってみると、この違いがわかるでしょう。

またこの表をよく見るとPOLL, ISET, RBYTEなど通常のN88-BASICでは使われていないキーワードがあることがわかります。これらはIEEEインターフェース用に用意されたものなのですが、IEEEを使うつもりがなければユーザー拡張命令用のキーワードとして使用できます。そのやり方は「第15章ランダムテクニック」をご覧ください。

* IEEEインターフェイスとは：計測器の相互接続に使われる規格で、正式にはIEEE488とかIECバスとか呼びます。ヒューレット・パッカード社が開発したので、HP-IBとかGP-IBともいいます。

(1)1バイトで表わされるもの

リスト 3-4

コード		キーワード
80	:	
81	:	END
82	:	FOR
83	:	NEXT
84	:	DATA
85	:	INPUT
86	:	DIM
87	:	READ
88	:	LET
89	:	GO TO
8A	:	RUN
8B	:	IF
8C	:	RESTORE
8D	:	GOSUB
8E	:	RETURN
8F	:	REM
90	:	STOP
91	:	PRINT
92	:	CLEAR
93	:	LIST
94	:	NEW
95	:	ON
96	:	WAIT
97	:	DEF
98	:	POKE
99	:	CONT
9A	:	OUT
9B	:	LPRINT
9C	:	LLIST
9D	:	CONSOLE
9E	:	WIDTH
9F	:	ELSE
A0	:	TRON
A1	:	TROFF
A2	:	SWAP
A3	:	ERASE
A4	:	EDIT
A5	:	ERROR
A6	:	RESUME
A7	:	DELETE
A8	:	AUTO
A9	:	RENUM
AA	:	DEFSTR
AB	:	DEFINT
AC	:	DEFSNG
AD	:	DEFDBL
AE	:	LINE
AF	:	WHILE
B0	:	WEND
B1	:	CALL
B2	:	
B3	:	
B4	:	
B5	:	WRITE
B6	:	COMMON
B7	:	CHAIN
B8	:	OPTION
B9	:	RANDOMIZE
BA	:	DSKO$
BB	:	OPEN
BC	:	FIELD
BD	:	GET
BE	:	PUT
BF	:	SET
C0	:	CLOSE
C1	:	LOAD
C2	:	MERGE
C3	:	FILES
C4	:	NAME
C5	:	KILL
C6	:	LSET
C7	:	RSET
C8	:	SAVE
C9	:	LFILES
CA	:	MON
CB	:	COLOR
CC	:	CIRCLE
CD	:	COPY
CE	:	CLS
CF	:	PSET
D0	:	PRESET
D1	:	PAINT
D2	:	TERM
D3	:	SCREEN
D4	:	BLOAD
D5	:	BSAVE
D6	:	LOCATE
D7	:	BEEP
D8	:	ROLL
D9	:	HELP
DA	:	
DB	:	KANJI
DC	:	TO
DD	:	THEN
DE	:	TAB(
DF	:	STEP
E0	:	USR
E1	:	FN
E2	:	SPC(
E3	:	NOT
E4	:	ERL
E5	:	ERR
E6	:	STRING$
E7	:	USING
E8	:	INSTR
E9	:	'
EA	:	VARPTR
EB	:	ATTR$
EC	:	DSKI$
ED	:	SRQ
EE	:	OFF
EF	:	INKEY$
F0	:	>
F1	:	=
F2	:	<
F3	:	+
F4	:	-
F5	:	*
F6	:	/
F7	:	^
F8	:	AND
F9	:	OR
FA	:	XOR
FB	:	EQV
FC	:	IMP
FD	:	MOD
FE	:	¥
FF	:	

(2) 2バイトで表わされるもの

リスト 3-5

```
FF 80 :
FF 81 : LEFT$
FF 82 : RIGHT$
FF 83 : MID$
FF 84 : SGN
FF 85 : INT
FF 86 : ABS
FF 87 : SQR
FF 88 : RND
FF 89 : SIN
FF 8A : LOG
FF 8B : EXP
FF 8C : COS
FF 8D : TAN
FF 8E : ATN
FF 8F : FRE
FF 90 : INP
FF 91 : POS
FF 92 : LEN
FF 93 : STR$
FF 94 : VAL
FF 95 : ASC
FF 96 : CHR$
FF 97 : PEEK
FF 98 : SPACE$
FF 99 : OCT$
FF 9A : HEX$
FF 9B : LPOS
FF 9C : CINT
FF 9D : CSNG
FF 9E : CDBL
FF 9F : FIX
FF A0 : CVI
FF A1 : CVS
FF A2 : CVD
FF A3 : EOF
FF A4 : LOC
FF A5 : LOF
FF A6 : FPOS
FF A7 : MKI$
FF A8 : MKS$
FF A9 : MKD$
FF AA :
FF AB :
FF AC :
FF AD :
FF AE :
FF AF :
FF B0 :
FF B1 :
FF B2 :
FF B3 :
FF B4 :
FF B5 :
FF B6 :
FF B7 :
FF B8 :
FF B9 :
FF BA :
FF BB :
FF BC :
FF BD :
FF BE :
FF BF :
FF C0 :
FF C1 :
FF C2 :
FF C3 :
FF C4 :
FF C5 :
FF C6 :
FF C7 :
FF C8 :
FF C9 :
FF CA :
FF CB :
FF CC :
FF CD :
FF CE :
FF CF :
FF D0 : DSKF
FF D1 : VIEW
FF D2 : WINDOW
FF D3 : POINT
FF D4 : CSRLIN
FF D5 : MAP
FF D6 : SEARCH
FF D7 : MOTOR
FF D8 : PEN
FF D9 : DATE$
FF DA : COM
FF DB : KEY
FF DC : TIME$
FF DD : WBYTE
FF DE : RBYTE
FF DF : POLL
FF E0 : ISET
FF E1 : IEEE
FF E2 : IRESET
FF E3 : STATUS
FF E4 : CMD
FF E5 :
FF E6 :
FF E7 :
FF E8 :
FF E9 :
FF EA :
FF EB :
FF EC :
FF ED :
FF EE :
FF EF :
FF F0 :
FF F1 :
FF F2 :
FF F3 :
FF F4 :
FF F5 :
FF F6 :
FF F7 :
FF F8 :
FF F9 :
FF FA :
FF FB :
FF FC :
FF FD :
FF FE :
FF FF :
```

3-3-2　中間言語テーブル

中間言語とキーワードの対応表は、ROM内の6B8AH番地から6E95H番地に格納されています。このテーブルは、入力したプログラムを中間言語に変換してテキスト領域に格納するときや、逆にリストをとったりするときに使われるものです。

・中間言語テーブルの形成

中間言語はキーワードの1文字目によって、アルファベット順に分類されています。分類されたキーワードの格納アドレスを示すテーブルが、6B56H番地から6B89H番地にあります。A～Zの各頭文字に対して2バイトのアドレスが順にならんでいます。

リスト　3-6

```
A  ------  6B8A        N  ------  6D40
B  ------  6BA0        O  ------  6D4F
C  ------  6BAF        P  ------  6D68
D  ------  6C0E        Q  ------  6D97
E  ------  6C4A        R  ------  6D98
F  ------  6C6F        S  ------  6DDA
G  ------  6C89        T  ------  6E21
H  ------  6C9B        U  ------  6E41
I  ------  6CA4        V  ------  6E4A
J  ------  6CCE        W  ------  6E58
K  ------  6CCF        X  ------  6E7B
L  ------  6CDC        Y  ------  6E7F
M  ------  6D1C        Z  ------  6E80
```

つまり、Aを頭文字とするキーワードグループは6B8AH～6B9FHにあるということです。また、このテーブルを参照しなくても、各グループの間には、セパレータとして00が書き込まれていますので、初めから順に追っていけばデータがどのグループに属するかがわかります。

例

```
ATTR$                    BSAVE
54  54  52  A4  EB  [00]  53  41  56
```

次に、中間言語テーブルのデータ構造を見てみましょう。ここでもメモリ節約のため、いろいろな工夫がなされています。

各データは、キーワードと中間言語とから成り立っています。キーワードのデータは1文字目が省略され(1文字目でグループ分けしてあるので不要)、キーワードの最後を示すために、最後の文字データの最上位ビットを1にしてあります。また、中間言語コードについては、1バイトのものではそのままの形で、2バイトのもの(FF+××)は、最上位ビットを0にして1バイトで表わせるようにしてあります。

<中間言語コードが1バイト>

```
キーワード          A    U    T    O     ┌A8  中間言語コード
キャラクタコード    41   55   54   4F    ↓
                         ↓    ↓    ↓
テーブルデータ           55   54   CF    A8
```

＜中間言語コードが２バイト＞

キーワード	H	E	X	$	FF 9A 中間言語コード
キャラクタコード	48	45	58	24	
テーブルデータ		45	58	A4	1A

次のプログラムで、これらのデータ(キーワードと中間言語)を出力してみましょう。

リスト 3-7

```
100 '
110 '       N88-BASIC Key Words List  ( 1 )
120 '
130  KW.TBL=&H6B8A
140  TOP.WORD=ASC("A")
150 OPEN "scrn:" FOR OUTPUT AS#1
160  PRINT #1,
170 '
180 *SEARCH.KW
190  KEY.WORD$=CHR$(TOP.WORD)
200 *NEXT.CODE
210  GOSUB *GET.CODE
220  IF CODE=0 THEN TOP.WORD=TOP.WORD+1 : GOTO *SEARCH.KW
230  IF CODE<&H80 THEN KEY.WORD$=KEY.WORD$+CHR$(CODE) : GOTO *NEXT.CODE
240 '
250  KEY.WORD$=KEY.WORD$+CHR$(CODE AND &H7F)
260  IF TOP.WORD=ASC("Z")+1 THEN KEY.WORD$=MID$(KEY.WORD$,2)
270  PRINT#1, TAB(10);KEY.WORD$;
280 '
290  GOSUB *GET.CODE
300  PRINT#1, TAB(20);
310  IF CODE<&H80 THEN PRINT#1, "FF ";
320  PRINT#1, RIGHT$("0"+HEX$(CODE OR &H80),2)
330 '
340  IF KW.TBL<&H6E95 THEN *SEARCH.KW
350 '
360  END
370 '
380 *GET.CODE
390  CODE=PEEK(KW.TBL)
400  KW.TBL=KW.TBL+1
410 RETURN
```

リスト　3-8

```
AUTO       A8
AND        F8
ABS        FF 86
ATN        FF 8E
ASC        FF 95
ATTR$      EB
BSAVE      D5
BLOAD      D4
BEEP       D7
CONSOLE    9D
COPY       CD
CLOSE      C0
CONT       99
CLEAR      92
CSRLIN     FF D4
CINT       FF 9C
CSNG       FF 9D
CDBL       FF 9E
CVI        FF A0
CVS        FF A1
CVD        FF A2
COS        FF 8C
CHR$       FF 96
CALL       B1
COMMON     B6
CHAIN      B7
COM        FF DA
CIRCLE     CC
COLOR      CB
CLS        CE
CMD        FF E4
DELETE     A7
DATA       84
DIM        86
DEFSTR     AA
DEFINT     AB
DEFSNG     AC
DEFDBL     AD
DSKO$      BA
DEF        97
DSKI$      EC
DSKF       FF D0
DATE$      FF D9
ELSE       9F
END        81
ERASE      A3
EDIT       A4
ERROR      A5
ERL        E4
ERR        E5
EXP        FF 8B
EOF        FF A3
EQV        FB
FOR        82
FIELD      BC
FILES      C3
FN         E1
FRE        FF 8F
FIX        FF 9F
FPOS       FF A6
GOTO       89
GO TO      89
GOSUB      8D
GET        BD
HEX$       FF 9A
HELP       D9
INPUT      85
ISET       FF E0
IEEE       FF E1
IRESET     FF E2
IF         8B
INSTR      E8
INT        FF 85
INP        FF 90
IMP        FC
INKEY$     EF
KEY        FF DB
KILL       C5
KANJI      DB
LOCATE     D6
LPRINT     9B
LLIST      9C
LPOS       FF 9B
LET        88
LINE       AE
LOAD       C1
LSET       C6
LIST       93
LFILES     C9
LOG        FF 8A
LOC        FF A4
LEN        FF 92
LEFT$      FF 81
LOF        FF A5
MOTOR      FF D7
MERGE      C2
MOD        FD
MKI$       FF A7
MKS$       FF A8
MKD$       FF A9
MID$       FF 83
MON        CA
MAP        FF D5
NEXT       83
NAME       C4
NEW        94
NOT        E3
OPEN       BB
OUT        9A
ON         95
OR         F9
OCT$       FF 99
OPTION     B8
OFF        EE
PRINT      91
PUT        BE
POKE       98
POLL       FF DF
POS        FF 91
PEEK       FF 97
PSET       CF
PRESET     D0
POINT      FF D3
PAINT      D1
PEN        FF D8
RETURN     8E
READ       87
RUN        8A
RESTORE    8C
RBYTE      FF DE
REM        8F
RESUME     A6
RSET       C7
RIGHT$     FF 82
RND        FF 88
RENUM      A9
RANDOMIZE  B9
ROLL       D8
SCREEN     D3
SEARCH     FF D6
STOP       90
SWAP       A2
SET        BF
SRQ        ED
STATUS     FF E3
SAVE       C8
SPC(       E2
STEP       DF
SGN        FF 84
SQR        FF 87
SIN        FF 89
STR$       FF 93
STRING$    E6
SPACE$     FF 98
THEN       DD
TRON       A0
TROFF      A1
TAB(       DE
TO         DC
TAN        FF 8D
TERM       D2
TIME$      FF DC
USING      E7
USR        E0
VAL        FF 94
VIEW       FF D1
VARPTR     EA
WIDTH      9E
WINDOW     FF D2
WAIT       96
WHILE      AF
WEND       B0
WRITE      B5
WBYTE      FF DD
XOR        FA
+          F3
-          F4
*          F5
/          F6
^          F7
¥          FE
'          E9
>          F0
=          F1
<          F2
```

また、前のプログラムを多少変更すると、中間言語コード順に並べ換えた表が得られます。

リスト　3-9

```
100 '
110 '      N88-BASIC Key Words List  ( 2 )
120 '
130  DIM K.W$(255)
140 OPEN "scrn:" FOR OUTPUT AS#1
150 '
160  KW.TBL=&H6B8A
170  TOP.WORD=ASC("A")
180 '
190 *SEARCH.KW
200  KEY.WORD$=CHR$(TOP.WORD)
210 *NEXT.CODE
220  GOSUB *GET.CODE
230  IF CODE=0 THEN TOP.WORD=TOP.WORD+1 : GOTO *SEARCH.KW
240  IF CODE<&H80 THEN KEY.WORD$=KEY.WORD$+CHR$(CODE) : GOTO *NEXT.CODE
250 '
260  KEY.WORD$=KEY.WORD$+CHR$(CODE AND &H7F)
270 '
280  GOSUB *GET.CODE
290  IF TOP.WORD=ASC("Z")+1 THEN KEY.WORD$=MID$(KEY.WORD$,2)
300  K.W$(CODE)=KEY.WORD$
310 '
320  IF KW.TBL<&H6E95 THEN *SEARCH.KW
330 '
340  PRINT #1,
350  PRINT#1, "----- STATEMENTS and COMMANDS -----"
360  FOR CODE=128 TO 255
370    PRINT#1, HEX$(CODE)" : ";K.W$(CODE)
380  NEXT
390  PRINT #1,
400  PRINT#1, "----- FUNCTIONS -----"
410  FOR CODE=0 TO 127
420    PRINT#1, "FF "HEX$(CODE+&H80)" : ";K.W$(CODE)
430  NEXT
440  PRINT #1,
450  END
460 '
470 *GET.CODE
480  CODE=PEEK(KW.TBL)
490  KW.TBL=KW.TBL+1
500  RETURN
```

3-3-4　リストでBEEP音を

中間言語の１から９は普通使われませんが、この中でおもしろい使い方ができるのが、コード7です。

これはBELコード(BEEP音)を表わし、直接キーボードから入力することはできません。

そこで、直接、中間コードを書き直すことにしましょう。

次のプログラムを入力します。('*'は、BEEP音を出したいところを示します)

リスト　3-10

```
list
10 REM *S*A*M*P*L*E*
Ok
```

次に、モニタモードで、必要な部分＊(コード2AHのところ)をコード7に直します。

リスト3-11

```
mon

h〕d8000,8016
8000 00 15 00 0A 00 8F 20 2A 53 2A 41 2A 4D 2A 50 2A
8010 4C 2A 45 2A 00 00 00
h〕s8007
8007 2A-07 53- 2A-07 41- 2A-07 4D- 2A-07 50- 2A-07 4C- 2A-07 45- 2A-07 00-
8015 00-
h〕d8000,8016
8000 00 15 00 0A 00 8F 20 07 53 07 41 07 4D 07 50 07
8010 4C 07 45 07 00 00 00
h〕.^b
Ok
```

BASICに戻して、リストをとってみて下さい。どうですか、1文字毎にBEEP音が出ますね。

リスト　3-12

```
list
10 REM SAMPLE
Ok
```

なお、この行を修正する(修正しなくても、この行のところで⏎キーを押す)と、リストをとっても音は出なくなります。

3-4　ラベルテーブル

ラベルテーブルは、ラベルテーブル先頭ポインタ(EB16,17H)で示されるアドレスから、単純変数テーブル先頭ポインタ(EB1B,1CH)で示されるアドレスの1つ前までです。

ラベルはこの中に、次のような形式で登録されます。

1バイト	2バイト	(ラベル名の長さ)バイト
ラベルの長さ	そのラベルがついている行の先頭アドレス−1	ラベル名

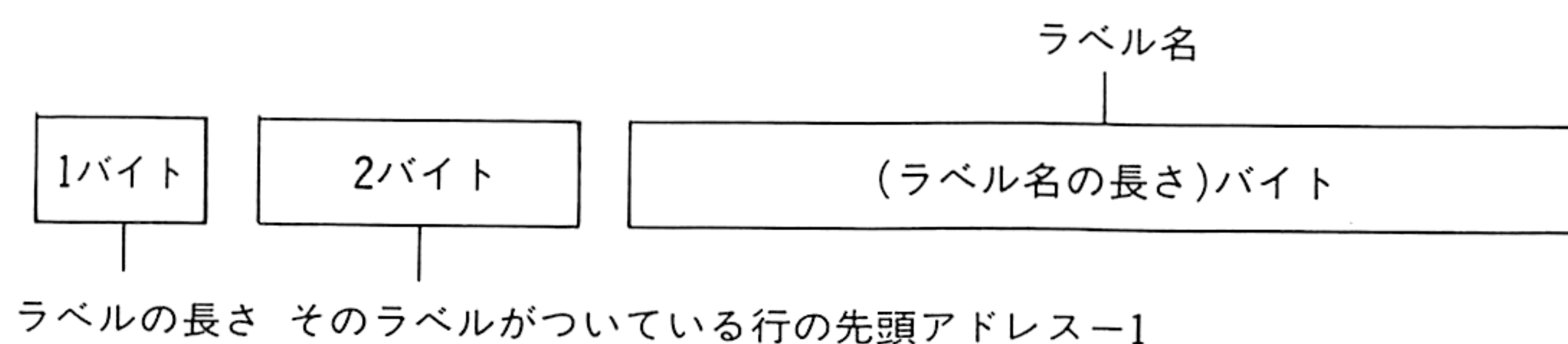



図3-4-A　ラベルの格納形式

それでは、実際にラベルがどのような形で登録されるか見てみましょう。

次のプログラムを入力した後、CLEAR文を実行します。

リスト　3-13

```
1000 *START
1010  GOSUB *INITIALIZE
1020 '
1030 *WRITE.BOX
1040  X=RND*600 : Y=RND*180
```

```
1050   CLR=INT(RND*7+1)
1060   LINE(X,Y)-STEP(40,20),CLR,B
1070   PAINT(X+20,Y+10),INT(RND*7+1),CLR
1080 GOTO *WRITE.BOX
1090 '
1100 *INITIALIZE
1110  SCREEN 0,0
1120  RANDOMIZE
1130  CLS 3
1140 RETURN
```

これでラベルテーブルができあがりました。各ポインタの値を見てみましょう。

EB16　E1　B2　　（ラベルテーブルの先頭）

EB1B　02　B3　　（単純変数テーブルの先頭）

ラベルテーブルは、B2E1H番地からB301H番地にあるわけですね。それではそちらの方をプログラムと比較して見てみます。

・ラベル・テーブル

ラベル長=5
ラベルのついている行の先頭アドレス−1=0
ラベル名="START"

```
B2B1  05 00 00 53 54 41 52 54 09 26 00 57 52 49 54 45
B2C1  2E 42 4F 58 0A B6 00 49 4E 49 54 49 41 4C 49 5A
B2D1  45
```

図3-4-B　ラベルテーブルの内容

・テキストウィンドウから見たプログラムデータ(実際は0000H番地から)

```
8000  00 0C 00 E8 03 F5 53 54 41 52 54 00 1F 00 F2 03
8010  20 8D 20 F5 49 4E 49 54 49 41 4C 49 5A 45 00 27
8020  00 FC 03 3A 8F E9 00 36 00 06 04 F5 57 52 49 54
8030  45 2E 42 4F 58 00 4E 00 10 04 20 58 F1 FF 88 F5
8040  1C 58 02 20 3A 20 59 F1 FF 88 F5 0F B4 00 62 00
8050  1A 04 20 43 4C 52 F1 FF 85 28 FF 88 F5 18 F3 12
8060  29 00 7D 00 24 04 20 AE 28 58 2C 59 29 F4 DF 28
8070  0F 28 2C 0F 14 29 2C 43 4C 52 2C 42 00 9E 00 2E
8080  04 20 D1 28 58 F3 0F 14 2C 59 F3 0F 0A 29 2C FF
8090  85 28 FF 88 F5 18 F3 12 29 2C 43 4C 52 00 AF 00
80A0  38 04 89 20 F5 57 52 49 54 45 2E 42 4F 58 00 B7
80B0  00 42 04 3A 8F E9 00 C7 00 4C 04 F5 49 4E 49 54
80C0  49 41 4C 49 5A 45 00 D2 00 56 04 20 D3 20 11 2C
80D0  11 00 D9 00 60 04 20 B9 00 E2 00 6A 04 20 CE 20
80E0  14 00 E8 00 74 04 8E 00 00 00 00 00 00 00 00 00
```

図3-4-C　プログラム中のラベル

3-5　変数テーブル

3-5-1　単純変数テーブル

単純変数テーブルは、ラベルテーブルの後に作られます。

この領域は、単純変数テーブル先頭ポインタ(EB1B,1CH)で示されるアドレスから、配列変数テーブル先頭ポインタ(EB1D,1EH)で示されるアドレスの1つ前までです。

プログラム中で変数が使われると、その型に応じた形式で、それが使われた順番に登録されていきます。変数の値の参照は、このテーブルの先頭から行なわれていきますので、頻繁に使う変数を早めに定義しておくと、実行速度を上げることができるわけです。各変数は、その型によって次のような形式で登録されます。

整数型　| 2 | 変数名* | データ |
▲(2バイト)

単精度型　| 4 | 変数名* | データ |
▲(4バイト)

倍精度型　| 8 | 変数名* | データ |
▲(8バイト)

文字型　| 3 | 変数名* | ストリングディスクリプタ** |
▲(3バイト)

(▲ VARPTRで示される)

*) 変数名

残りの変数名の長さ

変数の頭2文字
(1文字の場合，2バイト目は0)

残りの変数名
(最上位bitが1になっている)

**) ストリングディスクリプタ

文字列格納アドレス
文字列の長さ

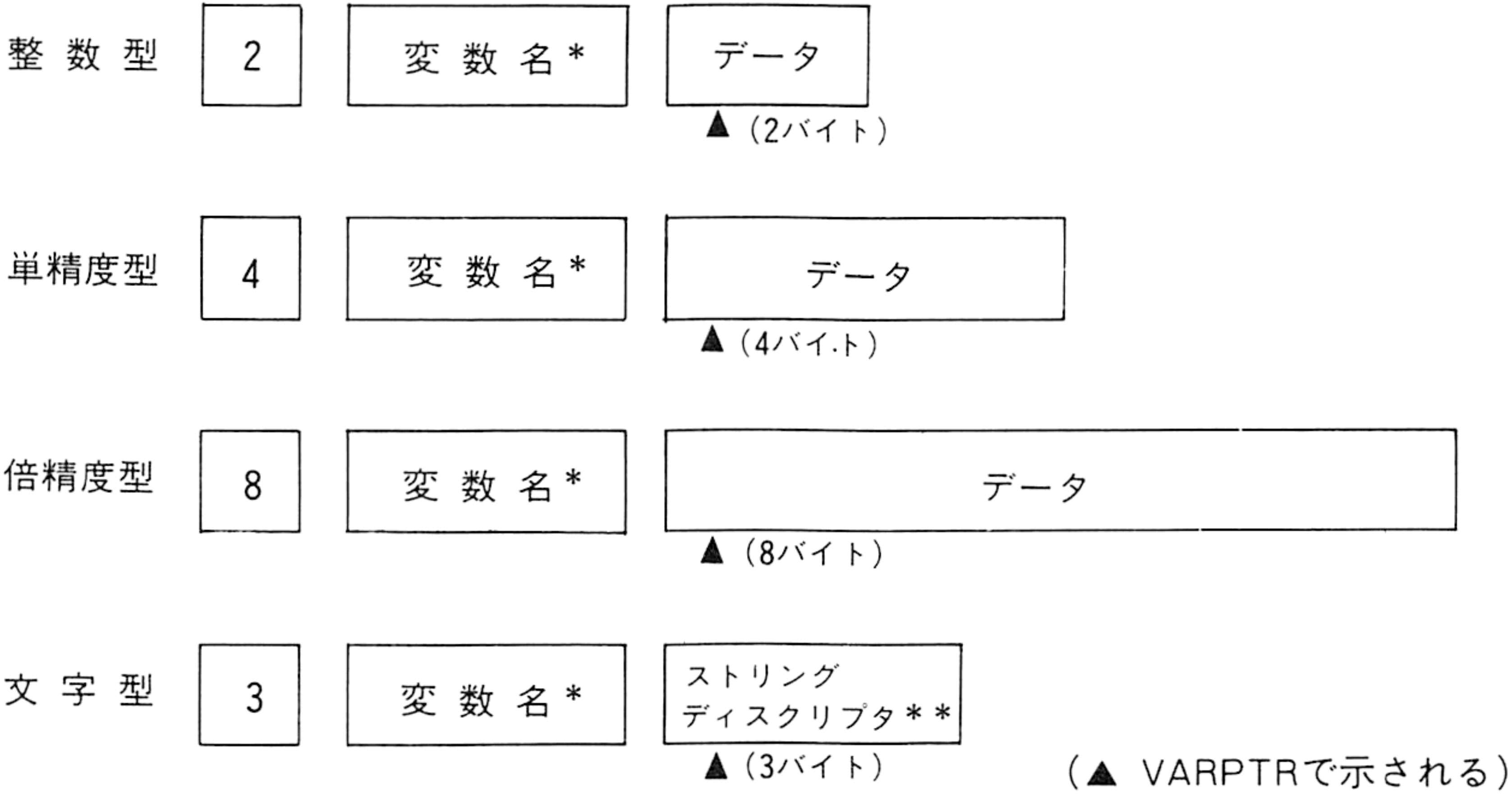
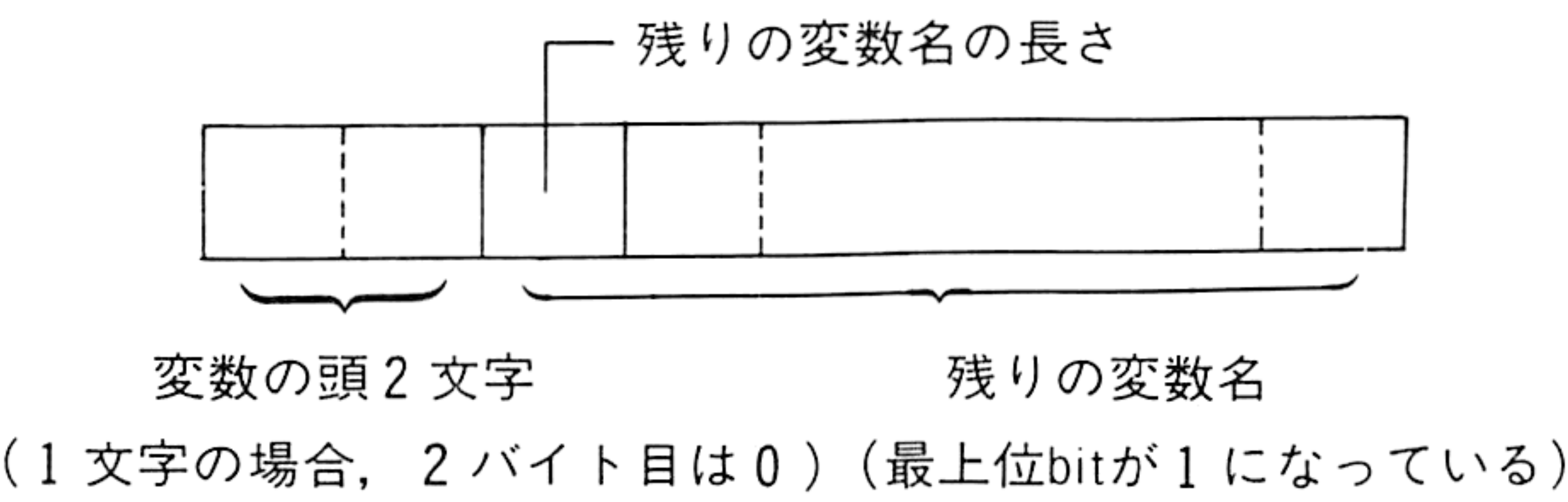
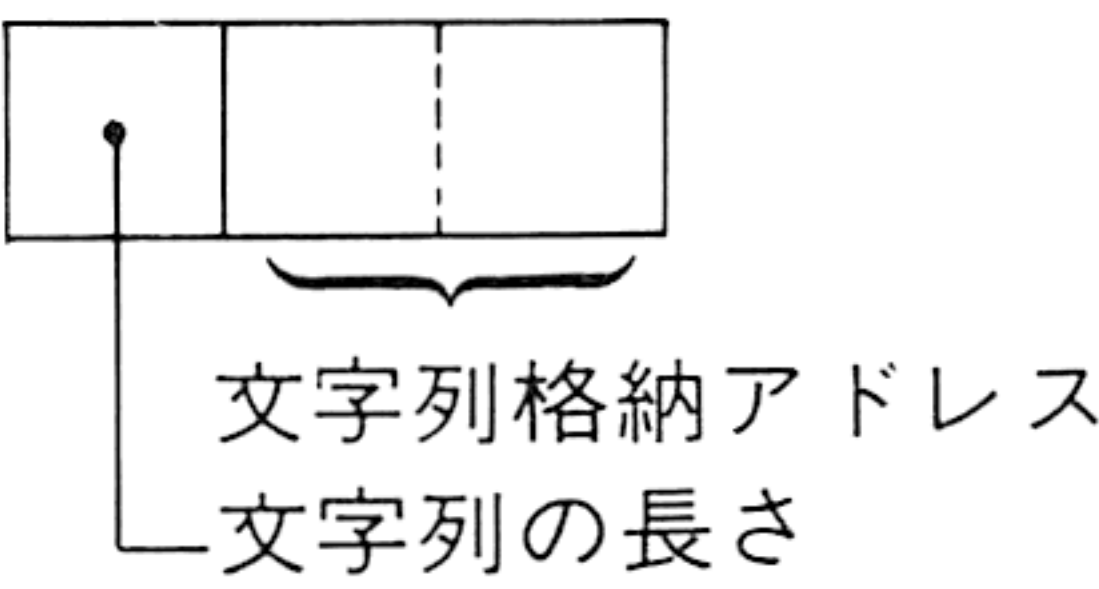



図3-5-A　単純変数テーブルの構造

実際の例で確かめてみます。次のプログラムを実行した後、各ポインタ及び単純変数テーブルを見てみましょう。

```
100 A%=1234
110 BCD%=-1234
120 DEFGHI!=1.2345
130 JKLMNO#=1.234567890123456#
140 PQRSTU$="1234567890"
```

※ポインタの値

```
h)deb1b
EB1B B1 B2 E5 B2
```

配列エリアの先頭　フリーエリアの先頭

※単純変数テーブル

```
h)db2b1,b2e4
B2B1 02 41 00 00 D2 04 02 42 43 01 C4 2E FB 04 44 45
B2C1 04 C6 C7 C8 C9 18 04 1E 81 08 4A 4B 04 CC CD CE
B2D1 CF C7 CF 62 14 52 06 1E 81 03 50 51 04 D2 D3 D4
B2E1 D5 0A F6 E3
```

図3-5-B　単純変数テーブルの例

ちょっとわかりにくいですね。各変数ごとに見てみましょう。

・A%=1234

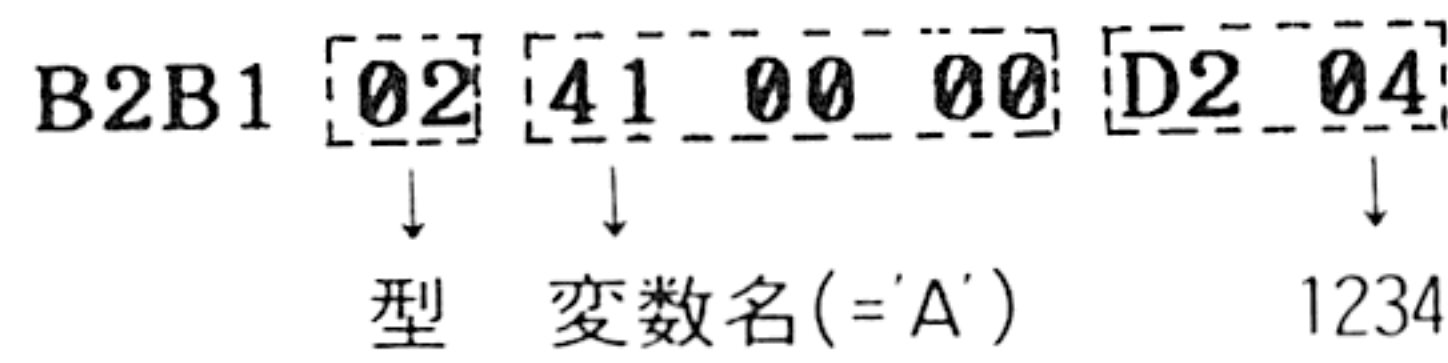
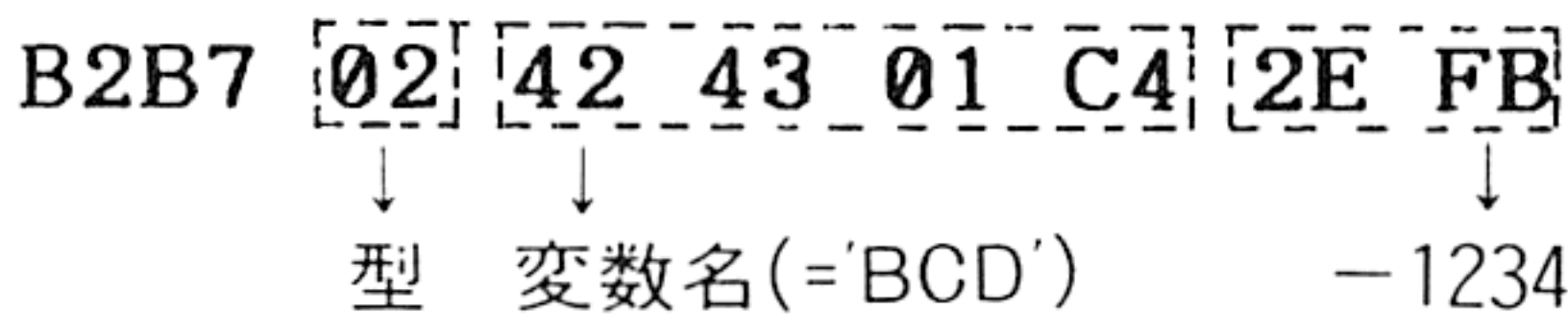
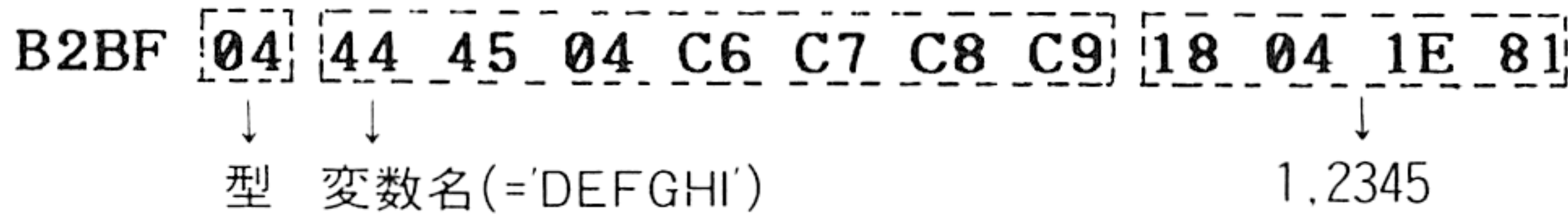
```
B2B1 [02] [41 00 00] [D2 04]
       ↓    ↓          ↓
       型   変数名(='A')  1234
```

・BCD%=−1234

```
B2B7 [02] [42 43 01 C4] [2E FB]
       ↓    ↓              ↓
       型   変数名(='BCD')   −1234
```

・DEFGH!=1.2345

```
B2BF [04] [44 45 04 C6 C7 C8 C9] [18 04 1E 81]
       ↓    ↓                           ↓
       型   変数名(='DEFGHI')          1.2345
```

・JKLMNO#=1.234567890123456#

```
B2CA [08] [4A 4B 04 CC CD CE CF] [C7 CF 62 14 52 06 1E 81]
       ↓    ↓                                 ↓
       型   変数名('JKLMNO')          1.1234567890123456
```

・PQRSTU$="1234567890"

```
B2DA [03] [50 51 04 D2 D3 D4 D5] [0A F6 E3]
       ↓    ↓                          ↓
       型   変数名(='PQRSTU')     ストリング・ディスクリプタ [文字列の長さ=10
                                                          文字列の格納アドレス
                                                              =E3F4H]
```

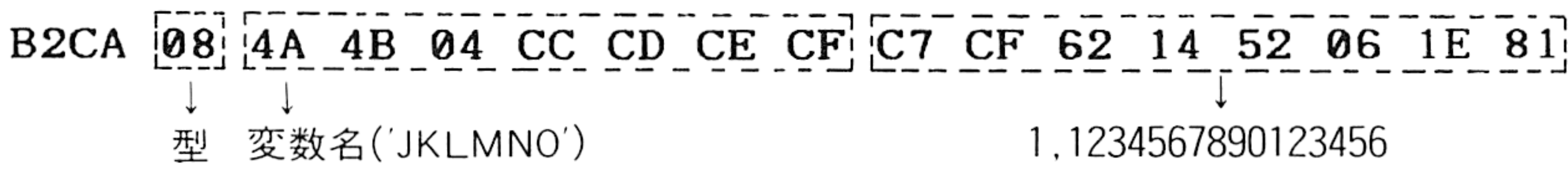
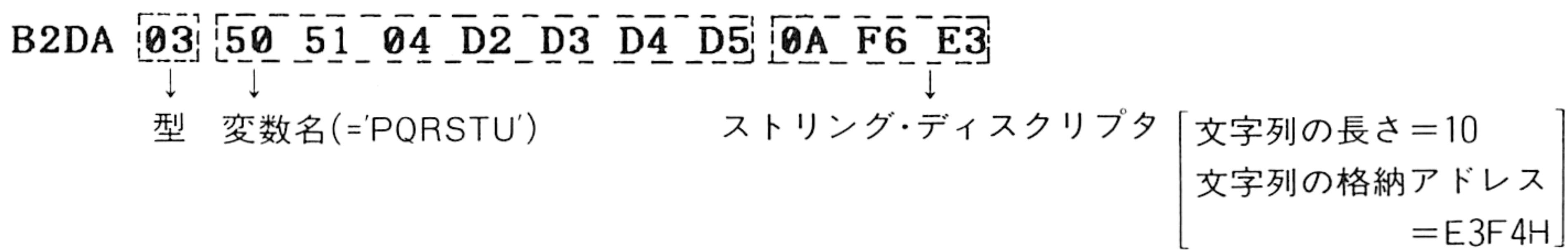



図3-5-C　変数ごとの例

3-5-2　配列変数テーブル

配列変数テーブルは単純変数テーブルの後に作られます。

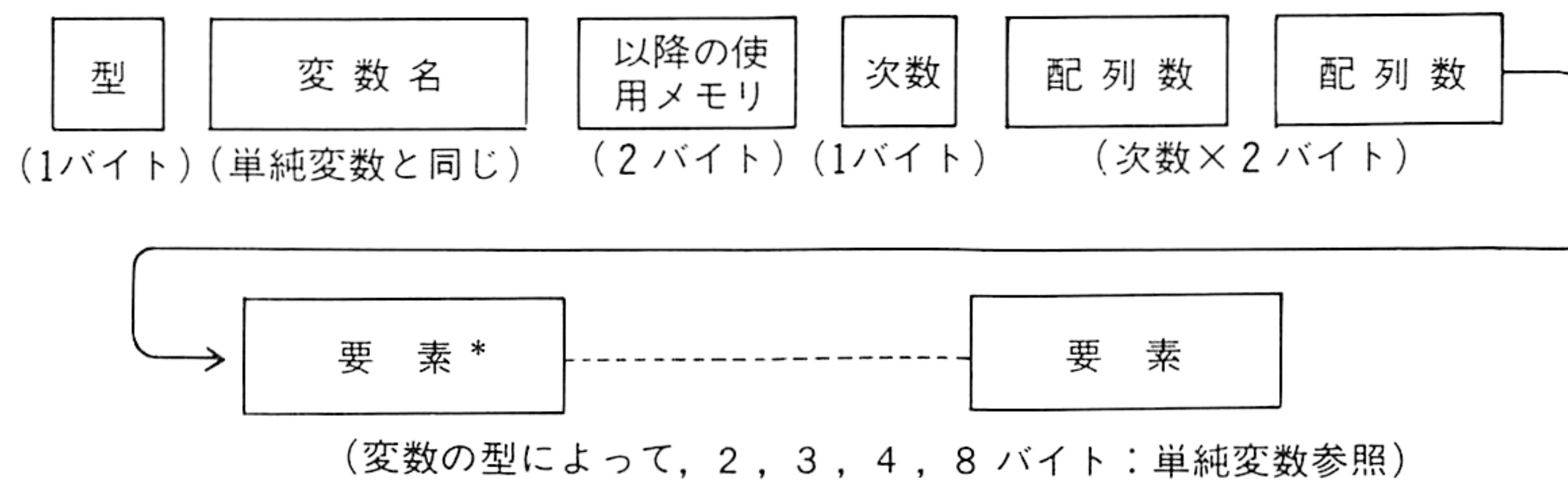
このテーブルは、配列変数テーブル先頭ポインタ(EB1D,1EH)で示されるアドレスから、フリースペース先頭ポインタ(EB1F,20H)で示されるアドレスの1つ前までです。

ここも、単純変数テーブルと同じように、DIM文を実行したり、添字が10以下の配列変数を使ったときに、その順番で登録されます。ERASE文を実行すると、その配列変数のテーブルが消されて、後ろにあるテーブルが前に移動してきます。

配列変数テーブルの形は、各配列について次のようになります。

型	変数名	以降の使用メモリ	次数	配列数	配列数
(1バイト)	(単純変数と同じ)	(2バイト)	(1バイト)	(次数×2バイト)	

→ | 要素* | ---------- | 要素 |

(変数の型によって，2，3，4，8バイト：単純変数参照)

*）要素の順番は　DIM A（2、3、4）の場合、次のようになる。
(OPTION BASE 0 場合)

A(0, 0, 0)
A(1, 0, 0)
A(2, 0, 0)
A(0, 1, 0)
A(1, 1, 0)
⋮
A(1, 3, 4)
A(2, 3, 4)

注）配列の宣言と逆順になる

図3-5-D　配列の格納形式

整数型配列変数と文字型配列変数について実際に見てみましょう。

●整数型配列変数

```
100 DIM A%(3,2)
110 FOR I=0 TO 3
120   FOR J=0 TO 2
130     A%(I,J)=I+J
140   NEXT J
150 NEXT I
```

※ポインタの値

```
h) deb1d
EB1D C1 B2 E4 B2
```

配列エリアの先頭　フリーエリアの先頭

※配列変数テーブル

```
h) db2c1,b2e3
```

型　変数名　以降の使用メモリ　次数　配列数　要素　A% (0.0)

```
B2C1 02 41 00 00 1D 00 02 03 00 04 00 00 00 01 00 02
B2D1 00 03 00 01 00 02 00 03 00 04 00 02 00 03 00 04
B2E1 00 05 00
```

要素A% (3.2)

●文字型配列変数

```
100 DIM ABC$(3,2)
110 FOR I=0 TO 3
120   FOR J=0 TO 2
130     ABC$(I,J)="ABC"
140   NEXT J
150 NEXT I
```

※ポインタの値

```
h) deb1d
EB1D C1 B2 F1 B2
```

配列エリアの先頭　フリーエリアの先頭

※配列変数テーブル

```
h) db2c1,b2f0
B2C1 03 41 42 01 C3 29 00 02 03 00 04 00 03 FD E3 03
B2D1 F4 E3 03 EB E3 03 E2 E3 03 FA E3 03 F1 E3 03 E8
B2E1 E3 03 DF E3 03 F7 E3 03 EE E3 03 E5 E3 03 DC E3
```

型　変数名　以降の使用メモリ　次数　配列数　要素 ABCS(0,0)のストリングディスクリプタ　要素 ABCS(3,2)のストリングディスクリプタ

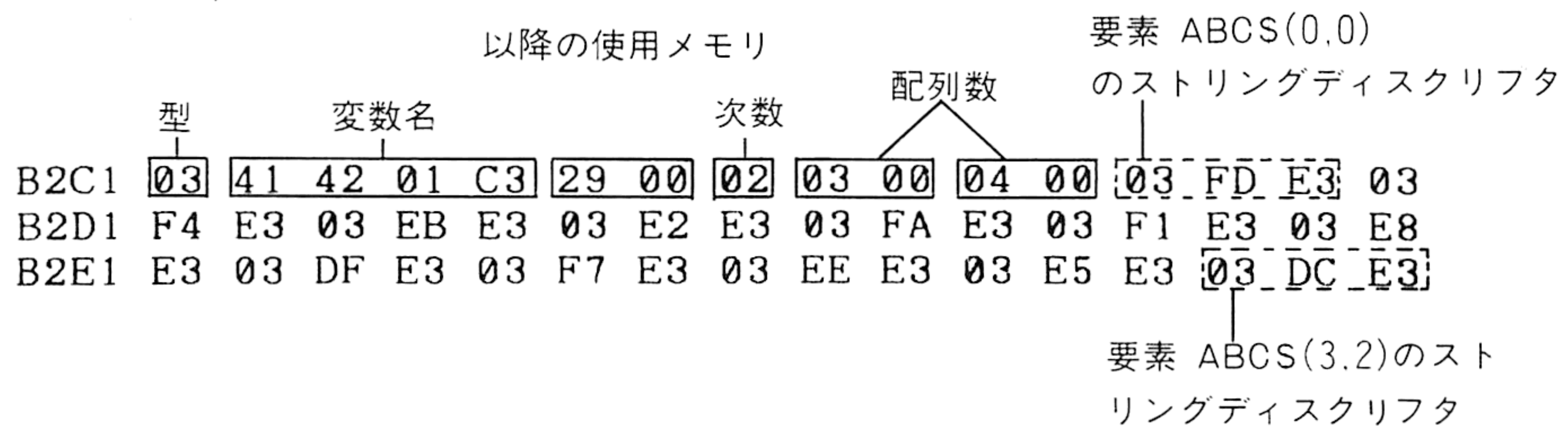



図3-5-E　整数型配列変数と文字型配列変数の例

3-6　文字列エリア

文字列エリアには文字型変数の実際の文字列が納められています。

例として次のプログラムを実行します。

リスト 3-14

```
100 A$="ABC"+"DE"
110 B$=CHR$(64)
120 A$="1234"
```

このとき文字列エリアは次のようになっています。

```
                  (EAF1,F2)                        (EACC,CD)
8000←                ↓                                ↓              →FFFF
─────────────────────────────────────────────────────┬──────────────────
                          1  2  3  4  @  A  B  C  D  E │ スタックエリア
─────────────────────────────────────────────────────┴──────────────────
                          ↑        ↑
A$のディスクリプタ      [4|アドレス]  │
                                    │
B$のディスクリプタ      [1|アドレス]─┘
                         長さ
```

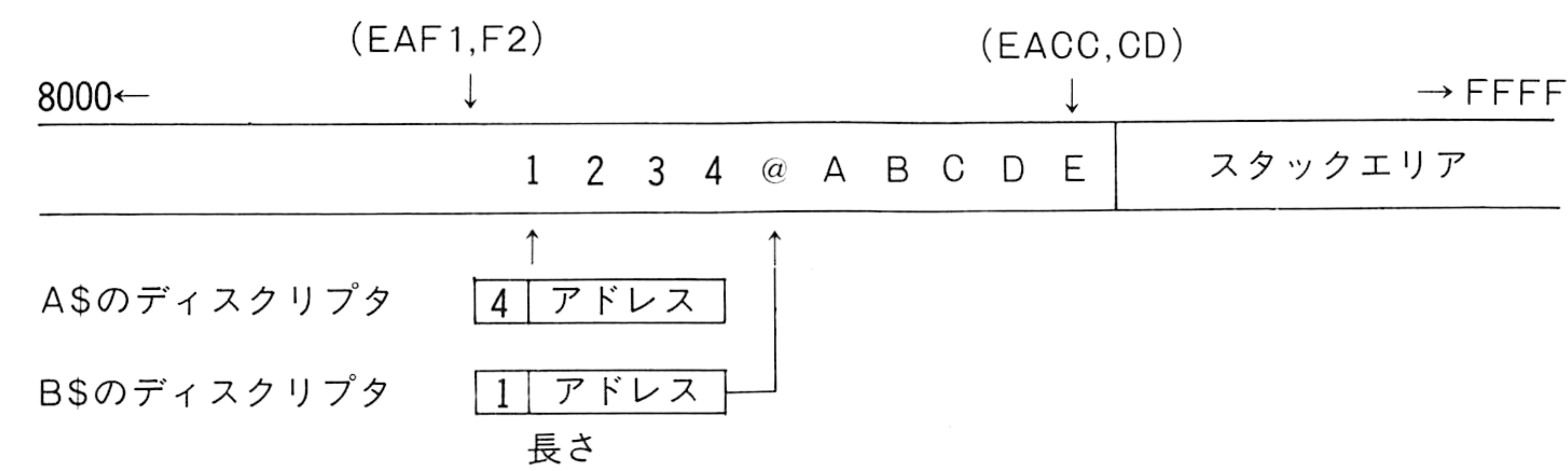



図 3-6-A　文字列エリアの様子(1)

「3-1-3 メモリ・マップの変化」で述べたとおり、最初にA$に代入された「ABCDE」はメモリ上から消えずにそのまま残っているのです。

ここでさらにダイレクトモードでA$="xyz" と行なうと次のようになります。

```
          (EAF1,F2)                                (EACC,CD)
8000←       ↓                                         ↓             →FFFF
─────────────────────────────────────────────────────┬─────────────────
                 x  y  z  1  2  3  4  @  A  B  C  D  E │ スタックエリア
─────────────────────────────────────────────────────┴─────────────────
```

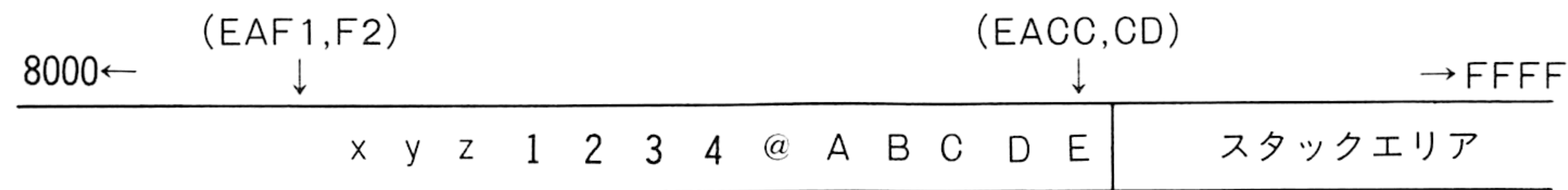



図 3-6-B　文字列エリアの様子(2)

ここでガーベジコレクションを行なってみましょう。

　　?FRE(0)

こうすると次のようになります。

```
                                  (EAF1,F2)  (EACC,CD)
8000←                                  ↓          ↓          →FFFF
──────────────────────────────────────────────────┬──────────────
             x  y  z  1  2  3  4  @  A  X  Y  Z  @ │ スタックエリア
──────────────────────────────────────────────────┴──────────────
                                       └──┬──┘   ↑
                                          ↑      B$
                                          A$
```

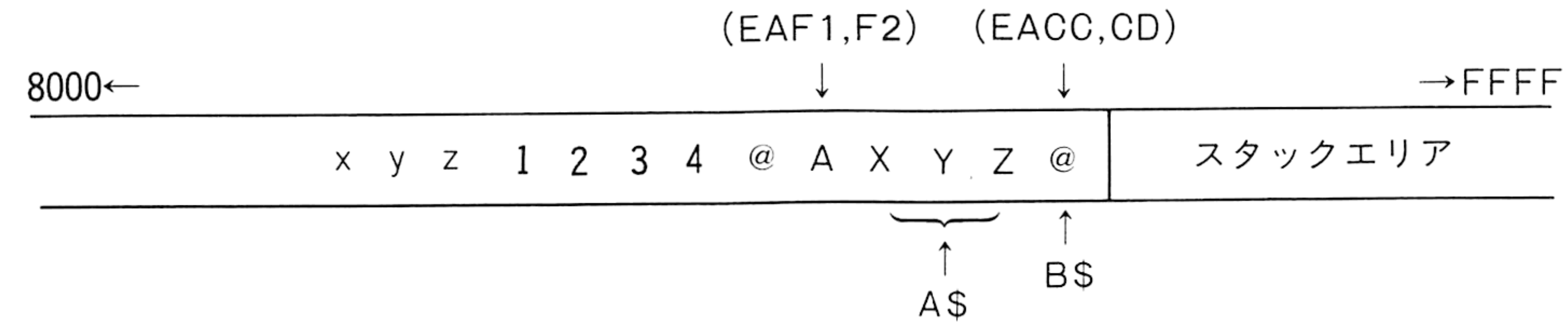



図 3-6-C　文字列エリアの様子(3)

現在使用されている「xyz」と「@」だけが、文字列エリアの後からつめられて、(EAF1, F2H)が移動しました。ここでB$="アイウ" +"エオ" と行なうと、このようになります。

```
                     (EAF1,F2)                        (EACC,CD)
8000←                    ↓                                ↓                    →FFFF
              x  y  z  1  アイウエオ  X  Y  Z  @  |  スタックエリア
                          B$            A$
```

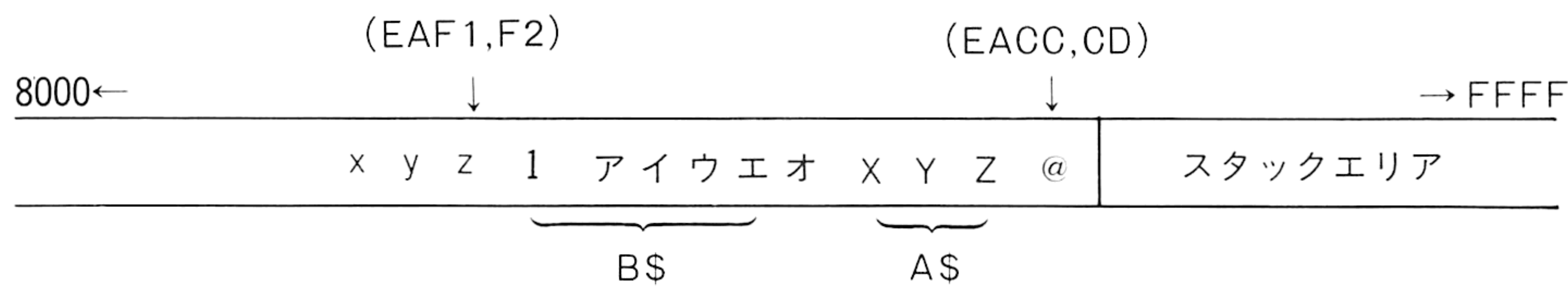



図3-6-D　文字列エリアの様子(4)

3-7 BASICプログラム復活

Newやリセットをした場合、以前入っていたプログラムは消えてしまうわけですが、実際にメモリから抹消されてしまうわけではなく、プログラムの最初のリンクポインタとプログラムの終わりを示すポインタ(EB18, 19H)がクリアされるだけです。従ってこれらの値を戻してやればプログラムが復活します。

```
0000 | 0 0          |                 | 0 0          |
     | リンクポインタ |   NEW           | 0 0          |
     |              |   リセット       | 0 0          |
     | プログラム    |   ⇒             |              |←(EB18,19)
     |              |                 | プログラム    |
     |              |                 | の残り        |
     | 00           |                 | 00           |
     | 00           |                 | 00           |
     |              |←(EB18,19)       |              |
7FFF |              |                 |              |

       最初のリンクポインタと(EB18,19)を戻す
```

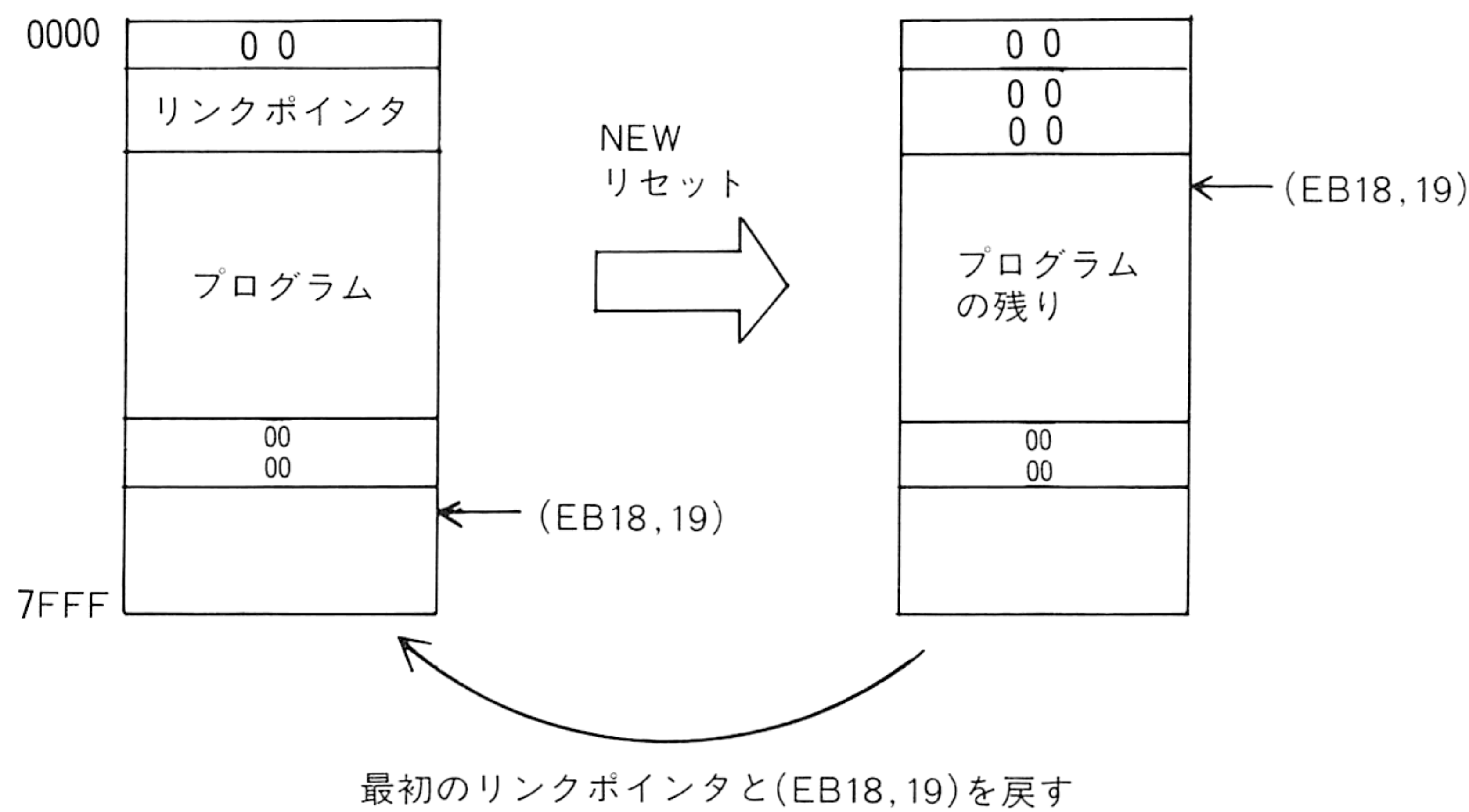



図3-7-A　復活の原理

これを自動的に行なうプログラムを次に紹介します。

リスト3-15

```
F260 2A 58 E6 36 01 CD BD 05 CD BB 1B CD D5 44 23 22
F270 18 EB AF 32 1A EB FF
```

newやリセットした直後にモニタに入り、上のプログラムを正確に打ち込みます。F260H番地から実行させると(GF260⏎)すぐに戻ってきますから、BASICに戻り、CLEAR⏎とします。これでプログラムが復活しました。ためしにLISTをとってみてください。

この方法は、ディスクを起動せずに作ったプログラムをディスクに入れる時にも使えます。リセットしてディスクを起動した後に、これを実行すればよいわけです。

第4章　入出力ファイル

4-1　入出力装置とファイル

N88-BASICでは、本体と入出力装置との情報のやりとりを「ファイル」という概念で行なっています。ファイルは本来ディスクなどの外部記憶装置に対して考え出された概念なのですが、現在ではもっと一般化されて、すべての入出力装置をファイルとしてとらえることが多くなっています。極端な例ではRAMファイルなどという主記憶の一部をファイルとして扱うものさえあります。(PC-8801mkII用CP／M(NEC製)では拡張RAMを仮想的なディスクとして扱っています。)

このようにすべての入出力装置をファイルとして統一的に扱うと、どの装置に対しても同じように相手をすることができ、入出力先の変更も容易にできます。

N88-BASICでは、ファイルは「デバイス名」と「ファイル名」で指定されます。ただし、ファイル名が存在するのは、一つのデバイスに複数のファイルを作ることができるディスクとカセットだけです。また、RS-232Cではファイル名の代わりに通信仕様を指定します。

4-2　変数でファイル指定

デバイス名とファイル名を合わせたものをファイルディスクリプタといい、両端をダブルクォーテーションで囲って表わします。つまり文字列になるわけですから、文字変数をファイルディスクリプタとして使うことができます。これにより入出力装置の変更がいっそう容易になります。

```
F$="2：DEMO"
```

として

```
OPEN F$ FOR OUTPUT AS ＃1
```

など、ファイルのアクセスをF$を使って行なえば、プログラム中から2：DEMOをすべて捜し出さなくても、F$=" LPT1：" としただけで、プリンタに出力先が変わります。この方法は次のようなときに使えます。

①プリンタに出力するプログラムをデバッグしているときに、いちいちプリンタへ出力していたのでは遅いし、紙の無駄にもなります。このようなときにF$="SCRN："としてデバッグし、あとでF$=" LPT1：" とすれば効率よく開発できます。

②結果をプリンタに出したいが時間がない、またはここにはプリンタがない、というときにはF$="ファイル名"としてディスクに出力しておき、あとからそれをプリンタに出します。プリンタへの打ち出しは、

```
10 OPEN "ファイル名" FOR INPUT AS ＃1
20 WHILE NOT EOF(1)
```

```
30        LPRINT  INPUT$(1,1)
40  WEND
50  CLOSE
```

というプログラムで簡単に行なえます。

③同じものを場合によって画面とプリンタへ切り換えて出力するときに、F$にSCRN：かLPT1：を入れておいて、F$でOPENして出力します。実際の例は「第11章プリンタ」で紹介します。

4-3 ファイルバッファ

N88-BASICではすべての入出力装置を統一的に扱うために、ファイルバッファというものが用意されています。ファイルバッファは特にディスクの入出力時に重要な働きをします。ディスクはハード的にはセクタ(N88-BASICでは256バイト)ごとにしか入出力できないからです。

右図のように＃0～＃nのファイルバッファがとられますが、各ファイルバッファの先頭アドレスを、ファイルポインタ（2バイト）が示しています。これはVARPTR（＃ファイル番号)で返されるものと同じです。

ファイルバッファは9バイトの作業領域(FCB)と256バイトのI／Oバッファから構成されています。

8000
テキストウィンドウ
ディスクコード
ドライブポインタ
ファイルポインタ＃0 ←(EC7F,80)が指す
ファイルポインタ＃1
ファイルポインタ＃2
⋮
ファイルポインタ＃n
ドライブテーブル
ファイルバッファ ＃0
ファイルバッファ ＃1
ファイルバッファ ＃2
⋮
ファイルバッファ ＃n ←VARPTR（＃n)
ラベルテーブル
FFFF
n:How many filesに対して入力した数

図4-3-A ファイルバッファ

8000 前のファイルバッファ
VARPTR(＃n)→ FCB 9バイト
VARPTR(＃n)+9→ I／O バッファ 256バイト
FFFF 次のファイルバッファ

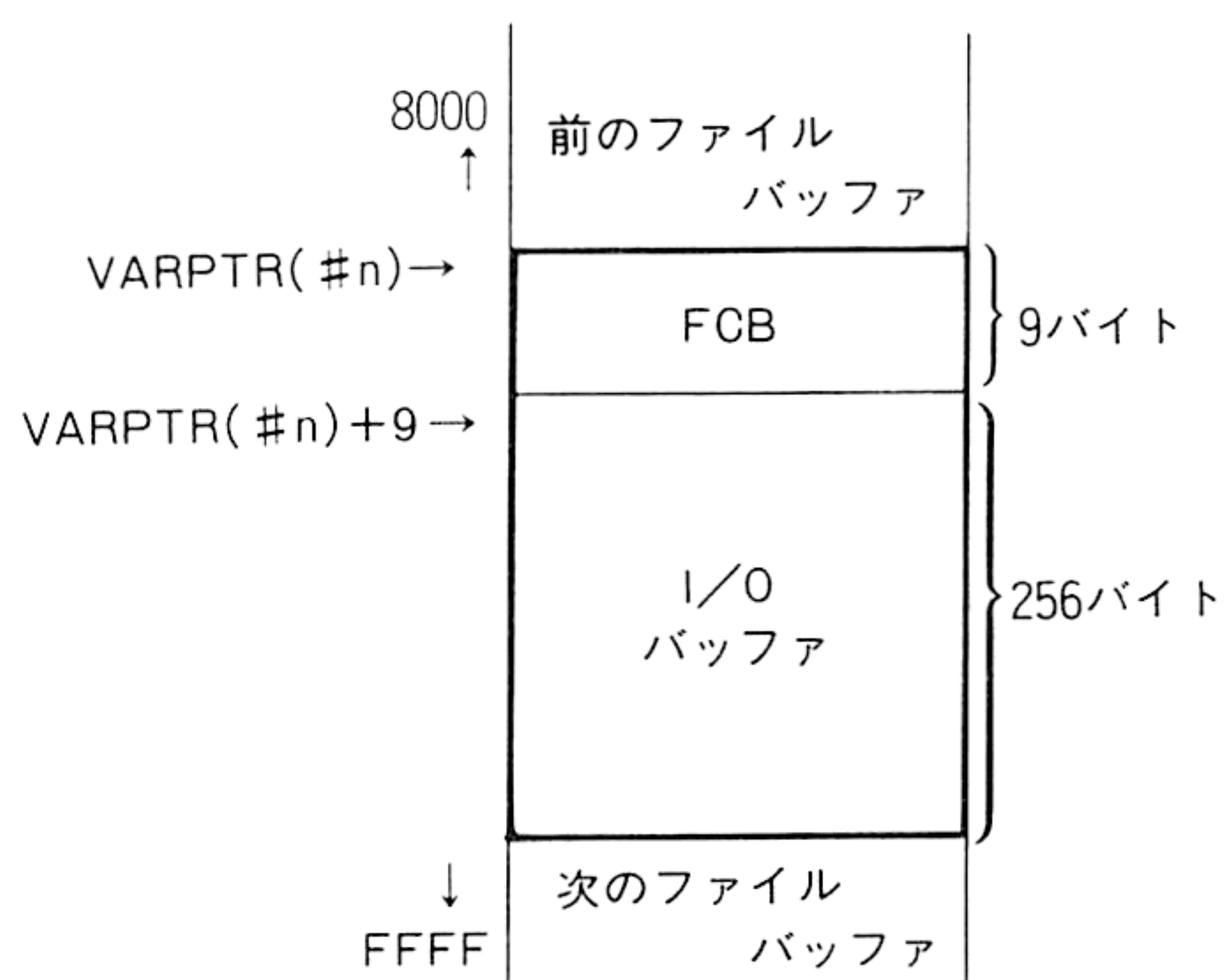
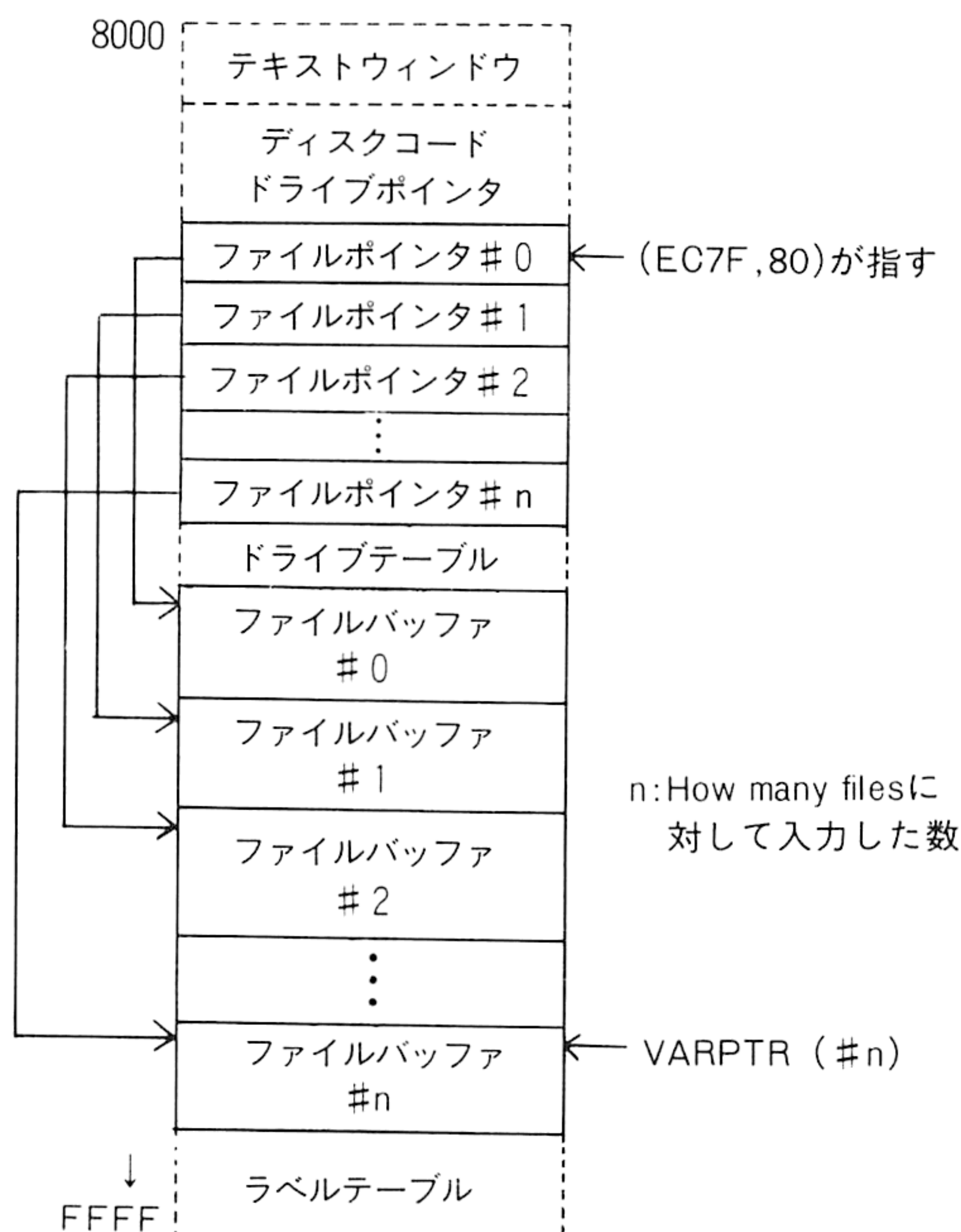



図4-3-B ファイルバッファの構成

作業領域の内分けは次の図表の通りです。

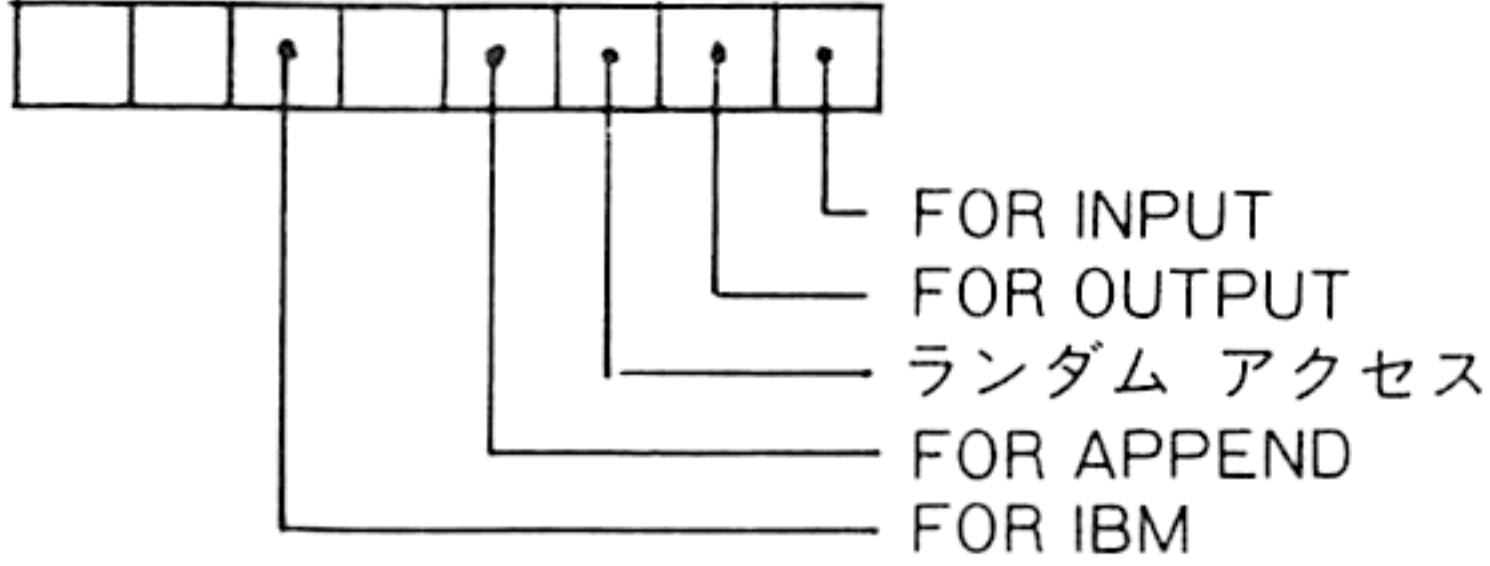
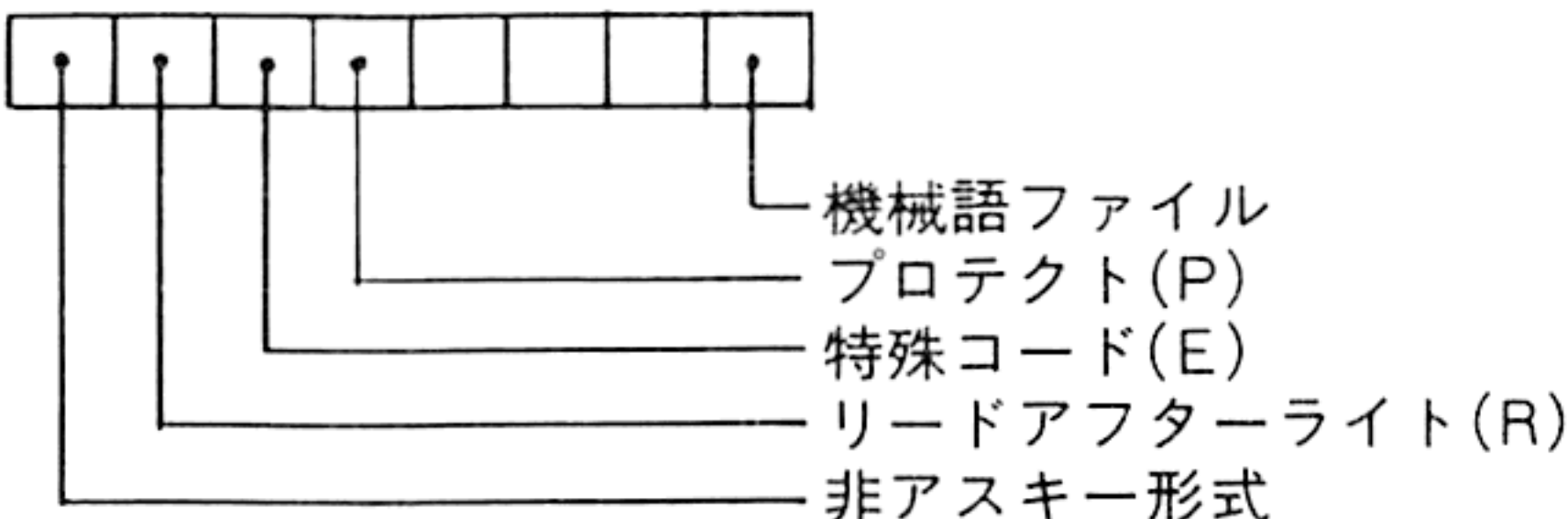
FCB（ファイル コントロール ブロック）

	項目	内容
VARPTR(♯n)	ファイルモード	ビット 7 6 5 4 3 2 1 0 ビット0: FOR INPUT ビット1: FOR OUTPUT ビット2: ランダム アクセス ビット3: FOR APPEND ビット5: FOR IBM CLOSEされている時は00
＋1	先頭クラスタ番号	ファイルが格納されている先頭のクラスタ番号（ディスクファイルのみ）
＋2	現在のクラスタ番号	アクセス中のクラスタ番号（ディスクファイルのみ）
＋3	現在のセクタ番号	セクタ番号
＋4	デバイス番号	下図参照
＋5	バッファ長	256バイトの時は00
＋6	データポインタ	シーケンシャルファイルの時、次に読み書きするデータの位置
＋7	ファイル属性	ビット 7 6 5 4 3 2 1 0 ビット0: 機械語ファイル ビット4: プロテクト(P) ビット5: 特殊コード(E) ビット6: リードアフターライト(R) ビット7: 非アスキー形式 CLOSEされている時は00
＋8	仮想ヘッド位置	シーケンシャルファイルで、レコード（CRで区切られる文字列）中でのデータポインタの位置

図4-3-C　FCB

デバイス番号	デバイス名	デバイス
0～7	1：～8：	ディスク（ドライブ番号－1）
F8	SCRN：	スクリーン
F9	LPT1：	プリンタ
FA	CAS2：	カセット600ボー
FB	CAS1：	カセット1200ボー
FC	COM3：	RS-232C チャネル3
FD	COM2：	〃 チャネル2
FE	COM1：	〃 チャネル1
FF	KYBD：	キーボード

図4-3-D　デバイス番号

◎ファイルバッファアドレス一覧表

2ドライブ時のファイルバッファアドレスの一覧表を載せておきます。これは [Aug 19, 1983] バージョンのものです。他のバージョンの場合(PC-8801のとき)や、ドライブ数が異なる場合は、下の出力プログラムを走らせてください。

リスト 4-1

```
100 '
110 '   FILE BUFFER ADDRESS TABLE
120 '
130  DEFINT A-Z
140 'FILTAB=&HAE1E : BIAS=-20620 :' (8801 KANJI BASIC May 15,1984)
150  FILTAB=&HAE3A : BIAS=-20592 :' (Aug 19,1983)
160 'FILTAB=&HADA6 : BIAS=-20740 :' (Apr 24,1982)
170 'FILTAB=&HAE1E : BIAS=-20620 :' (Aug 20,1982)
180 'FILTAB=&H8400 : BIAS=-31742 :' ROM  BASIC
190 '
200 OPEN "lpt1:" FOR OUTPUT AS#1
210 PRINT #1,CHR$(27)+"Q"+CHR$(27)+"T10"
220 '
230 '-------- OPEN FILE#
240 PRINT #1,"  # OF OPEN FILES  ";
250 FOR I=0 TO 15 : PRINT #1,USING" ##  ";I; : NEXT
260 NF=-1 : GOSUB *LF
270 '-------  FILE POINTERS ADRS
280 PRINT #1,"FILE POINTERS TOP  ";
290 FOR N=0 TO 15
300   PRINT #1,HEX$(FILTAB);" ";
310 NEXT
320 NF=-1 : GOSUB *LF
330 PRINT #1,"FILE POINTERS END  ";
340 FOR N=0 TO 15
350   PRINT #1,HEX$(FILTAB+N*2+1);" ";
360 NEXT
370 NF=-1 : GOSUB *LF
380 '
390 '-------- FILE BUFFERS
400 '    --  WORK  --
410 FOR MOF=0 TO 15
420   PRINT #1,"       F C B  TOP  ";
430   FOR N=0 TO MOF-1
440     PRINT #1,"     ";
450   NEXT
460   FOR N=MOF TO 15
470     VPTR=MOF*265+N*2+BIAS
480     PRINT #1,HEX$(VPTR);" ";
490   NEXT
500   NF=MOF : GOSUB *LF
510 '    --  TOP OF BUFFER  --
520   PRINT #1,"       BUFFER TOP  ";
530   FOR N=0 TO MOF-1
540     PRINT #1,"     ";
550   NEXT
560   FOR N=MOF TO 15
570     VPTR=MOF*265+N*2+BIAS
580     PRINT #1,HEX$(VPTR+9);" ";
590   NEXT
600   NF=-1 : GOSUB *LF
610 NEXT
620 '
630 PRINT #1,CHR$(27);"N";CHR$(27);"H";
640 PRINT #1,CHR$(27)+"A"
650 END
660 '
670 '----- line feed
680 *LF
690 PRINT #1,
700 IF NF>=0 THEN 730
```

```
710    PRINT #1,STRING$(99,"-")
720 RETURN
730    PRINT #1,USING" _###  ";NF;
740    PRINT #1,STRING$(93,"-")
750 RETURN
```

リスト　4-2

N88-DISK BASIC　　　2 ドライブ

ファイルバッファアドレス一覧表

ファイル番号	# OF OPEN FILES	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	FILE POINTERS TOP	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A	AE3A
	FILE POINTERS END	AE3B	AE3D	AE3F	AE41	AE43	AE45	AE47	AE49	AE4B	AE4D	AE4F	AE51	AE53	AE55	AE57	AE59
# 0	F C B TOP	AF90	AF92	AF94	AF96	AF98	AF9A	AF9C	AF9E	AFA0	AFA2	AFA4	AFA6	AFA8	AFAA	AFAC	AFAE
	BUFFER TOP	AF99	AF9B	AF9D	AF9F	AFA1	AFA3	AFA5	AFA7	AFA9	AFAB	AFAD	AFAF	AFB1	AFB3	AFB5	AFB7
# 1	F C B TOP		B09B	B09D	B09F	B0A1	B0A3	B0A5	B0A7	B0A9	B0AB	B0AD	B0AF	B0B1	B0B3	B0B5	B0B7
	BUFFER TOP		B0A4	B0A6	B0A8	B0AA	B0AC	B0AE	B0B0	B0B2	B0B4	B0B6	B0B8	B0BA	B0BC	B0BE	B0C0
# 2	F C B TOP			B1A6	B1A8	B1AA	B1AC	B1AE	B1B0	B1B2	B1B4	B1B6	B1B8	B1BA	B1BC	B1BE	B1C0
	BUFFER TOP			B1AF	B1B1	B1B3	B1B5	B1B7	B1B9	B1BB	B1BD	B1BF	B1C1	B1C3	B1C5	B1C7	B1C9
# 3	F C B TOP				B2B1	B2B3	B2B5	B2B7	B2B9	B2BB	B2BD	B2BF	B2C1	B2C3	B2C5	B2C7	B2C9
	BUFFER TOP				B2BA	B2BC	B2BE	B2C0	B2C2	B2C4	B2C6	B2C8	B2CA	B2CC	B2CE	B2D0	B2D2
# 4	F C B TOP					B3BC	B3BE	B3C0	B3C2	B3C4	B3C6	B3C8	B3CA	B3CC	B3CE	B3D0	B3D2
	BUFFER TOP					B3C5	B3C7	B3C9	B3CB	B3CD	B3CF	B3D1	B3D3	B3D5	B3D7	B3D9	B3DB
# 5	F C B TOP						B4C7	B4C9	B4CB	B4CD	B4CF	B4D1	B4D3	B4D5	B4D7	B4D9	B4DB
	BUFFER TOP						B4D0	B4D2	B4D4	B4D6	B4D8	B4DA	B4DC	B4DE	B4E0	B4E2	B4E4
# 6	F C B TOP							B5D2	B5D4	B5D6	B5D8	B5DA	B5DC	B5DE	B5E0	B5E2	B5E4
	BUFFER TOP							B5DB	B5DD	B5DF	B5E1	B5E3	B5E5	B5E7	B5E9	B5EB	B5ED
# 7	F C B TOP								B6DD	B6DF	B6E1	B6E3	B6E5	B6E7	B6E9	B6EB	B6ED
	BUFFER TOP								B6E6	B6E8	B6EA	B6EC	B6EE	B6F0	B6F2	B6F4	B6F6
# 8	F C B TOP									B7E8	B7EA	B7EC	B7EE	B7F0	B7F2	B7F4	B7F6
	BUFFER TOP									B7F1	B7F3	B7F5	B7F7	B7F9	B7FB	B7FD	B7FF
# 9	F C B TOP										B8F3	B8F5	B8F7	B8F9	B8FB	B8FD	B8FF
	BUFFER TOP										B8FC	B8FE	B900	B902	B904	B906	B908
#10	F C B TOP											B9FE	BA00	BA02	BA04	BA06	BA08
	BUFFER TOP											BA07	BA09	BA0B	BA0D	BA0F	BA11
#11	F C B TOP												BB09	BB0B	BB0D	BB0F	BB11
	BUFFER TOP												BB12	BB14	BB16	BB18	BB1A
#12	F C B TOP													BC14	BC16	BC18	BC1A
	BUFFER TOP													BC1D	BC1F	BC21	BC23
#13	F C B TOP														BD1F	BD21	BD23
	BUFFER TOP														BD28	BD2A	BD2C
#14	F C B TOP															BE2A	BE2C
	BUFFER TOP															BE33	BE35
#15	F C B TOP																BF35
	BUFFER TOP																BF3E

Aug 19, 1983 Version

データの入出力はファイルバッファを介して行なわれると書きましたが、実際にI／Oバッファを使うのは、ディスクの入出力とRS-232Cの入力の場合だけです。ディスク以外の出力はバッファを用いずに行なわれます。カセットからの入力はF0D3H～F152Hのカセット入力バッファを用い、キーボードからの入力はEFD9H～EFF8Hのキー入力バッファを用いています。

では実際にファイルバッファにどのように書き込まれていくか、見てみましょう。2ドライブ、How many files＝2, 使用するファイル番号＝＃1とします。このとき前ページの表よりFCB：B009H～B011H, I／Oバッファ：B012H～B111Hとなります。

リスト　4-3

```
open "FILE" for output as#1
Ok
mon

h) db09d,b0a5
B09D [02] [30] 30 [00] [00] [00] [00] [0]2 00
h) db0a6
B0A6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
h) ^b
Ok
```

先頭クラスタ＃（ディスケットによって異なります。）
デバイス＃（ドライブ1）
データポインタ（まだ何も書かれていない）
バッファ長＝256
属性なし
I／Oバッファはすべて00

次にデータを書き込みます。

リスト　4-4

```
print #1,"ABCDEF";
Ok
mon

h) db09d,b0a5
B09D 02 30 30 01 00 00 [06] 08 [06]
h) db0a6
B0A6|41 42 43 44 45 46 00 00 00 00 00 00 00 00 00 00
h) ^b  A  B  C  D  E  F
Ok  |──→レコード
```

データポインタ（6 文字書き込んだ）
まだ CRが現れない

リスト　4-5

```
print #1,"XYZ"
Ok
mon

h) db09d,b0a5
B09D 02 30 [30] [01] 00 00 [0B] 08 [01]
h) db0a6
B0A6|41 42 43 44 45 46 58 59 5A 0D|0A 00 00 00 00 00
h) ^b  A  B  C  D  E  F  X  Y  Z  CR|LF
Ok  |←──────── レコード ────────→|──→ レコード
```

クラスタ117
セクタ1
現在のI／Oバッファがクラスタ117, セクタ1に書き込まれるデータであることを示す

長い文字列を書き込みます。

リスト　4-6

```
print #1,string$(250,"@")
Ok      ここでディスクをアクセスしました。
mon                   バッファが一杯になったため、ディスクに書き込んで
                      次のセクタのデータをバッファしている。
h)db09d,b0a5
B09D 02 30 30 [02] 00 00 [07] 08 01
h)db0a6
B0A6 40 40 40 40 40 0D|0A 00 00 00 00 00 00 00 00 00
h)^b  @  @  @  @  @  CR |LF
Ok                    レコード ←─┼─→ レコード
```

CLOSEします。

リスト　4-7

```
close
Ok      ディスクへのアクセス
mon
                    CLOSEされていることを示します
h)db09d,b0a5
B09D [00] 30 30 02 00 00 07 [00] 01
h)db0a6
B0A6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
h)^b                    I/Oバッファはすべて00
Ok
```

今の例ではクラスタ117に書き込みました。クラスタ117は5インチ両面の場合はトラック29(1DH)、サーフェス0、セクタ9～16ですからモニタの^dコマンドで見てみます。

リスト　4-8

```
mon

h)^d1,0,c,1
Track 0C,Surface 00,Sector 01
0000 41 42 43 44 45 46 58 59 5A 0D 0A 40 40 40 40 40    ABCDEFXYZ..@@@@@
0010 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0020 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0030 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0040 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0050 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0060 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0070 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0080 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
0090 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
00A0 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
00B0 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
00C0 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
00D0 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
00E0 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
00F0 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40    @@@@@@@@@@@@@@@@
h)^d1,0,c,2
Track 0C,Surface 00,Sector 02
0000 40 40 40 40 40 0D 0A 1A 00 00 00 00 00 00 00 00    @@@@@...
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
h)^b
Ok
```

4-4　キュー

前節で、キー入力とカセット入力では専用のバッファを使うと述べましたが、このバッファはキューといわれるものの仲間です。これは最初に入れたデータを最初にとり出すもので、FIFO(First In First Out)とも呼ばれます。

下図のようなキューがあるとします。データとして、A, B, C, Dが入っています。

		A	B	C	D			

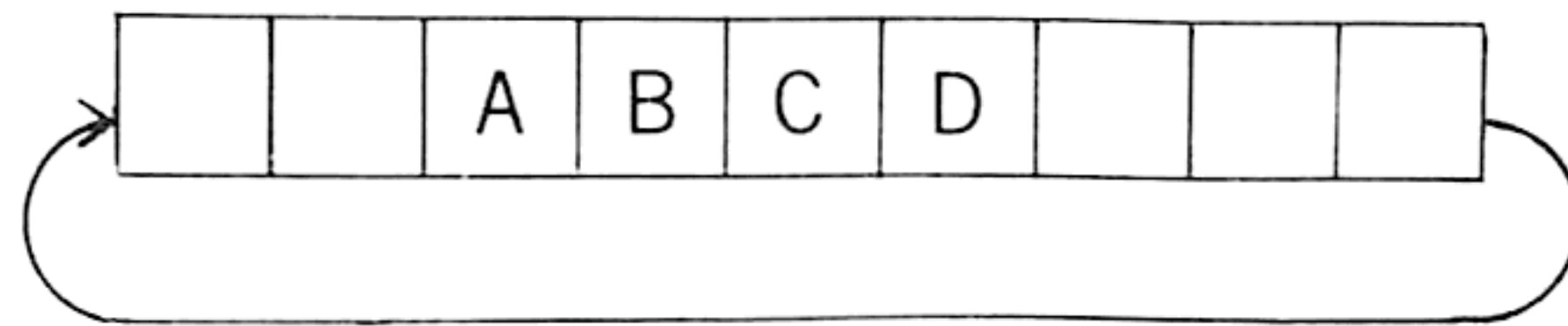



図4-4-A　キューの原理(1)

データE, F, Gを入れます。

		A	B	C	D	E	F	G

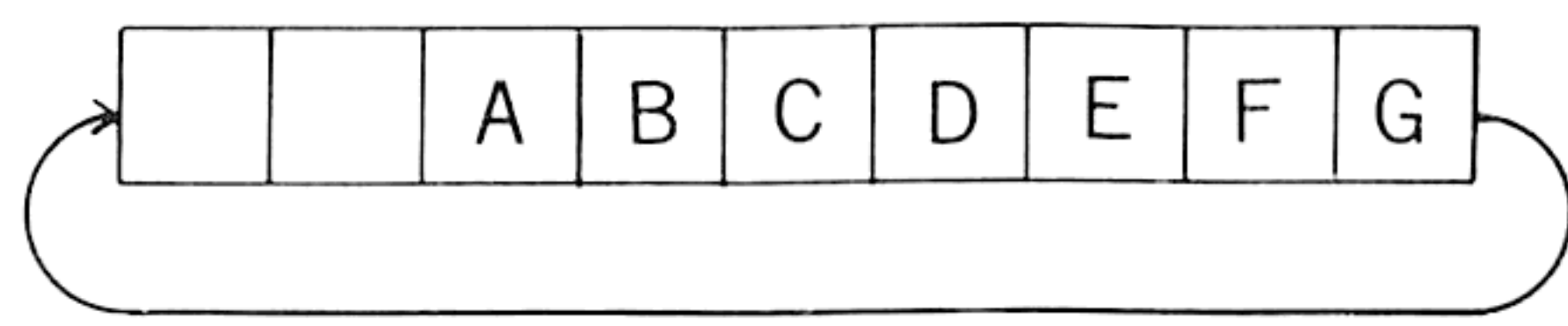



図4-4-B　キューの原理(2)

データA, Bを取り出します。

				C	D	E	F	G

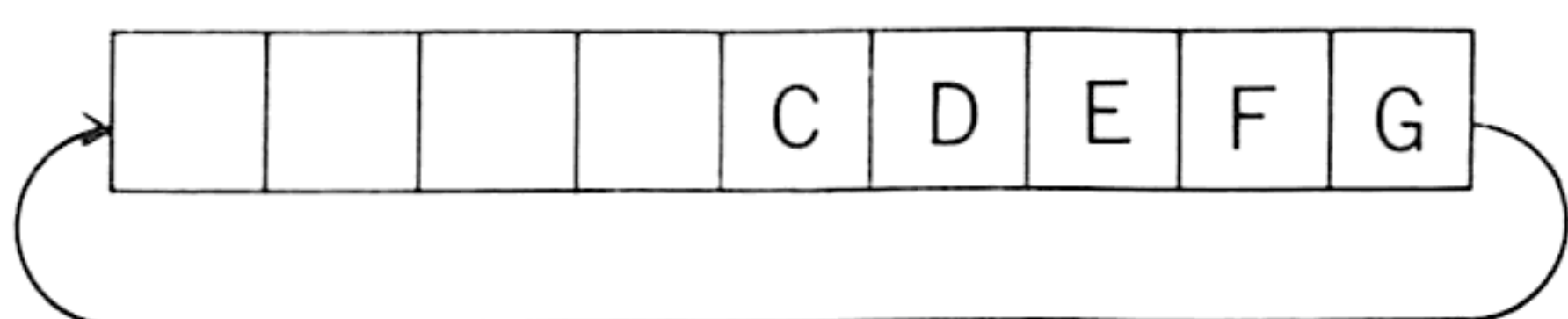



図4-4-C　キューの原理(3)

ここでデータHを入れるとこのようになります。

H				C	D	E	F	G

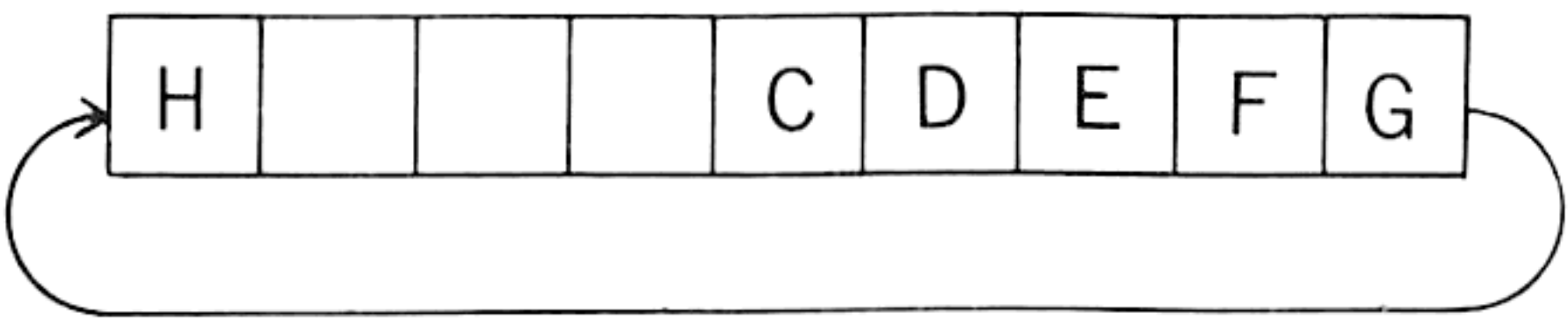



図4-4-D　キューの原理(4)

データC, Dをとり出し、データI, J, Kを入れます。

H	I	J	K			E	F	G

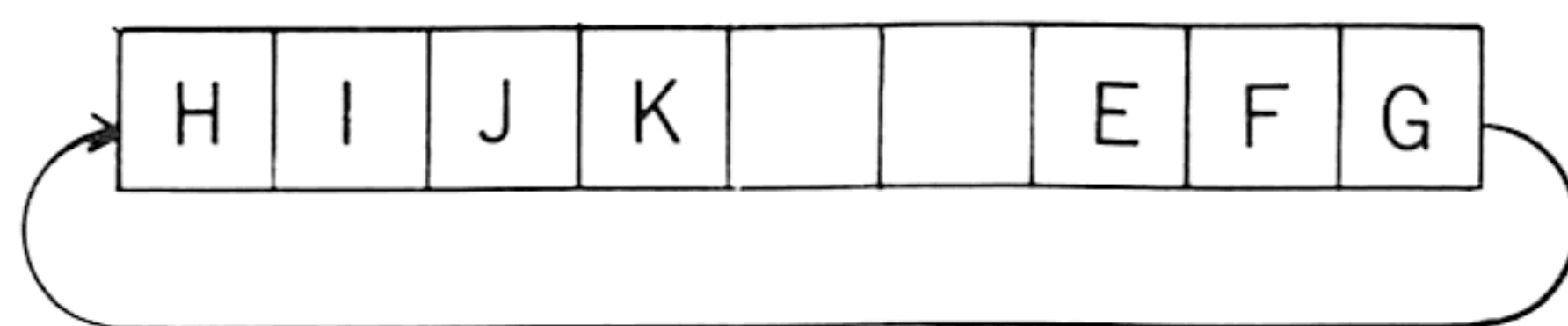



図4-4-E　キューの原理(5)

このようにしてデータの出し入れが行なわれますが、コンピュータがこの動作を行なうためにはいくつかの作業領域が必要です。

N$_{88}$-BASICでは作業領域として、次のものを持っています。

・空いている部分の最後(データの先頭-1)の位置

・データの最後(空き部分の先頭-1)の位置

・キューのアドレス

・キューの長さ

これらはキューテーブルとしてひとまとめにして格納されています。キューテーブルは6バイトで、次のようになっています。

キューテーブル先頭	プット　オフセット	データの最後（キューの先頭を0とした位置）
＋1	ゲット　オフセット	空いている部分の最後（キューの先頭を0とした位置）
＋2	バック　キャラクタ	キューから文字を取り出すルーチンで、ここにデータ（≠0）があるとその文字をキューからのデータとする
＋3	キュー長	キューの長さ 2^n-1、$1 \leqq n \leqq 8$
＋4，＋5	キューアドレス	キューのアドレス

キューアドレス
↓
キュー：［　｜データ｜　］
↑ゲットオフセット　↑プットオフセット
|←　キューの長さ＋1　→|

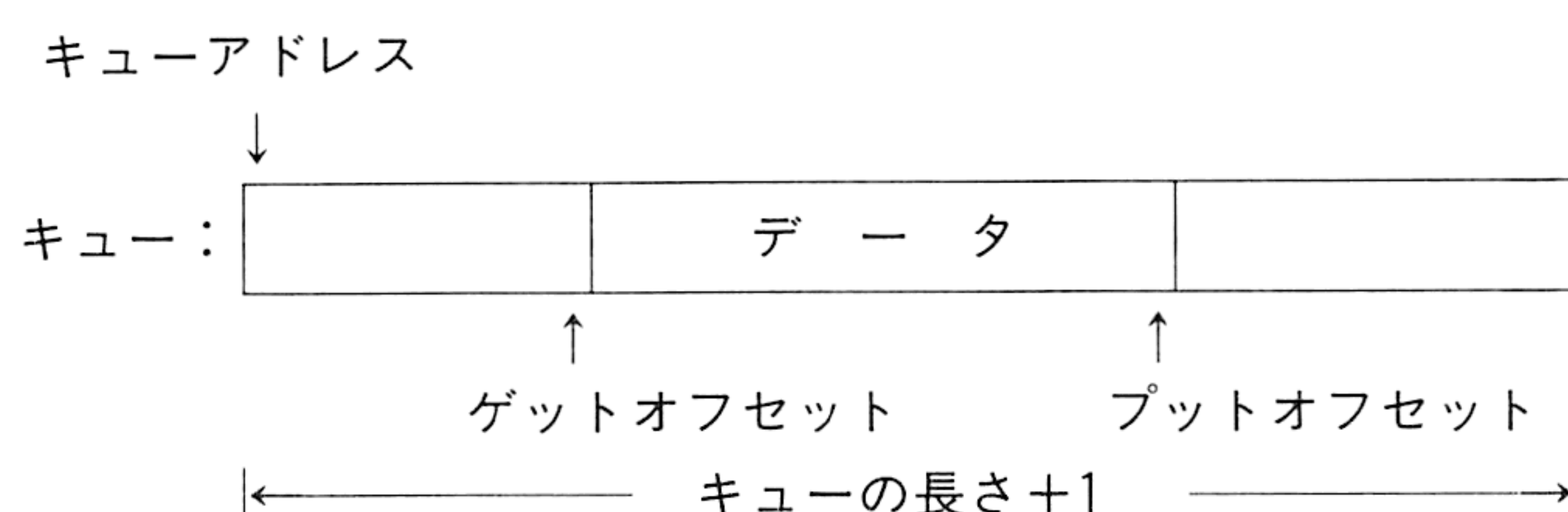



図4-4-F　キューテーブル

キューテーブルには＃0と＃1の2つがあり、＃0はキー入力用に、＃1はカセット入力とRS232C入力用に使われています。キューテーブル＃0のアドレスは(E6CB, CCH)に入っています。普通はEFCDHです。＃1は＃0のすぐ後ろにあります。

EFCD　キューテーブル ＃0　←(E6CB,CC)
EFD3　キューテーブル ＃1
EFD9　キー入力用キュー

キューテーブル＃0 → キー入力用キュー　EFD9～EFF8
キューテーブル＃1 → カセット入力用キュー　F0D3～F152
キューテーブル＃1 → RS-232C用キュー（ファイルバッファ）

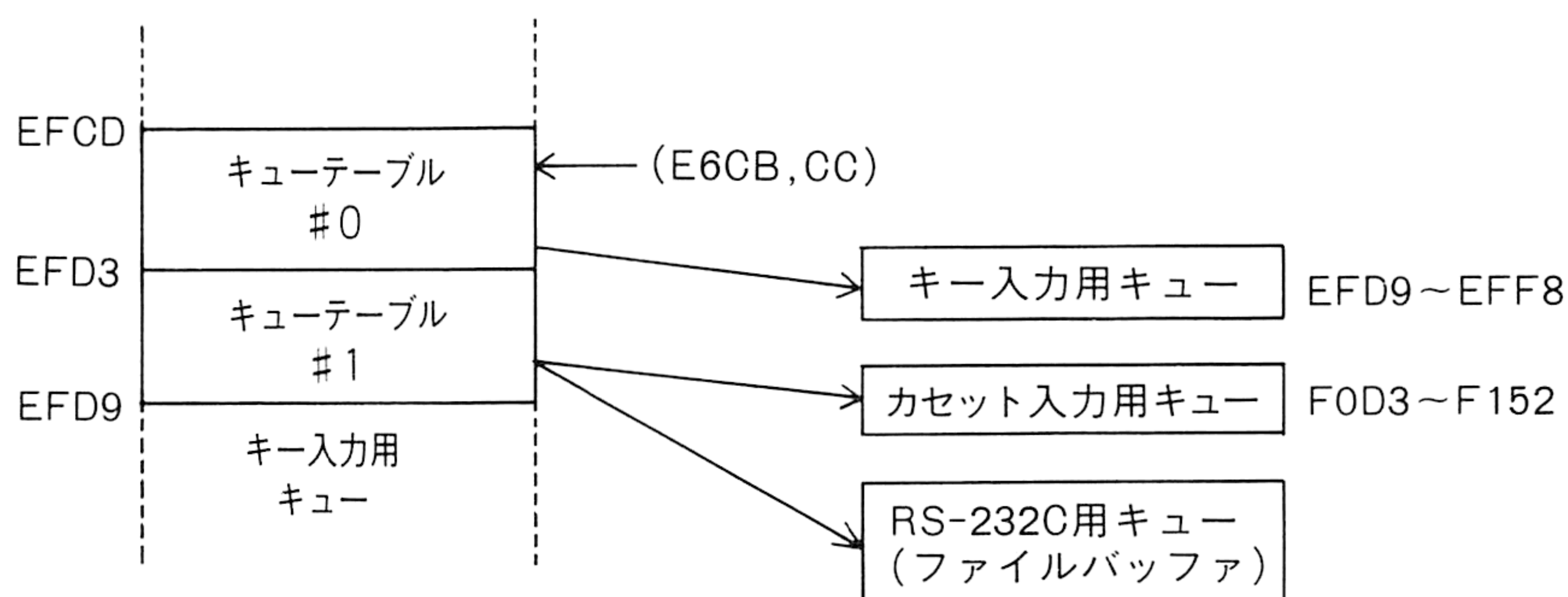



図4-4-G　キューの割り当て

このキューを利用すると、ファンクションキーのようなことができます。詳しくは「第5章キー入力」を見て下さい。

第5章　キー入力

N88-BASICのキー入力は次のような流れで行なわれます。

ハードウエア　キーボード → キーバッファ（メモリ） → INPUT文 / INKEY$関数 / INPUT$関数 など

キースキャン ← CPU → キーバッファ

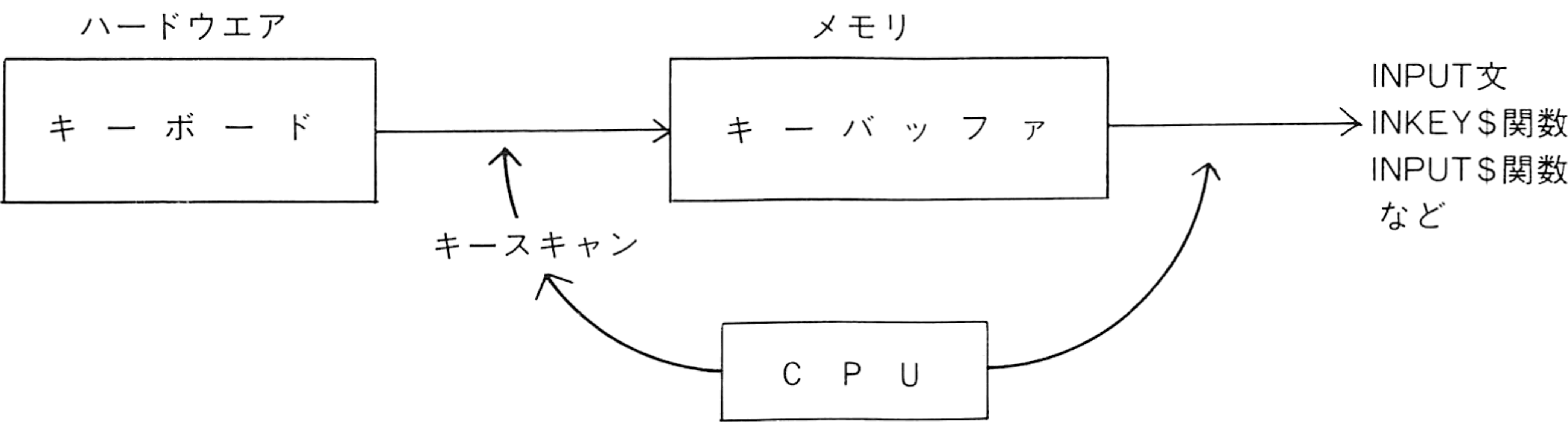
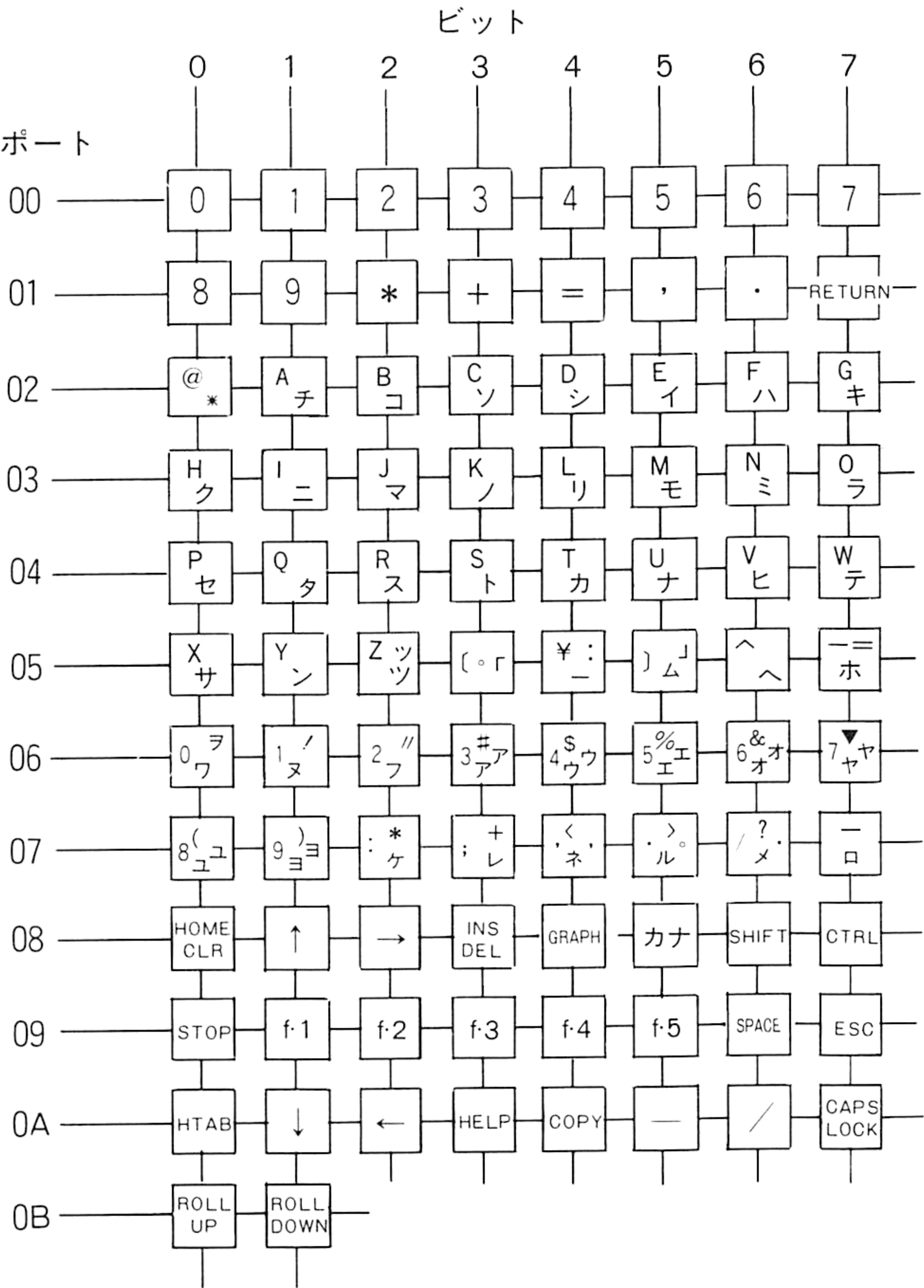



図5-A　キー入力の流れ

これらについて説明します。

5-1　キーボード

キーボードはCPUのバスにつながっていてIN命令(BASICではINP関数)で読み込むことができます。各キーは図5-1-Aのようにマトリックス状に配列されています。横の行にそれぞれI/Oアドレスが割り当てられていて、そのアドレスのポートを読むことにより、その行の8つのキー状態を知ることができます。

キーが押されていると、読んだデータの対応するビットが0になります。押されていないキーのビットは1になっています。

ポート＼ビット	0	1	2	3	4	5	6	7
00	0	1	2	3	4	5	6	7
01	8	9	*	+	=	,	.	RETURN
02	@ ゛	A チ	B コ	C ソ	D シ	E イ	F ハ	G キ
03	H ク	I ニ	J マ	K ノ	L リ	M モ	N ミ	O ラ
04	P セ	Q タ	R ス	S ト	T カ	U ナ	V ヒ	W テ
05	X サ	Y ン	Z ッ ツ	[° 「	¥ : ー	] ム 」	^ ヘ	- = ホ
06	0 ヲ ワ	1 ! ヌ	2 " フ	3 # ア ァ	4 $ ウ ゥ	5 % エ ェ	6 & オ ォ	7 ▼ ヤ ャ
07	8 (ユ ュ	9) ヨ ョ	: * ケ	; + レ	, < ネ 、	. > ル 。	/ ? メ ・	ー ロ
08	HOME CLR	↑	→	INS DEL	GRAPH	カナ	SHIFT	CTRL
09	STOP	f·1	f·2	f·3	f·4	f·5	SPACE	ESC
0A	HTAB	↓	←	HELP	COPY	—	/	CAPS LOCK
0B	ROLL UP	ROLL DOWN						

図5-1-A　キーマトリックス

たとえばAキーが押された時を考えてみましょう。他のキーは押されていないとします。Aキーはポート02Hにつながっています。したがって、Aキーの状態を知るには、ポート02Hの値を読みます。Aキーのビットはビット1に位置付けられています。ですから、ポート02Hは2進で11111101、つまり、FDHになっているはずです。

では、実際にやってみましょう。次のプログラムを入力して下さい。

```
10    CLS
20    LOCATE  0, 0
30    PRINT  HEX$(INP(02))
40    GOTO    20
```

これはポート02Hの値を画面の左上に表示し続けるものです。runさせるとFFと表示されています。

ここでAキーを押してみて下さい。表示がFDに変わりました。離すと、FFにもどります。押したり、離したりしてみて下さい。押している間だけFDになることが、おわかりになると思います。

では今度は、Bキーを押して下さい。FBが表示されましたね。Bキーは同じポート02Hでもビット2に位置付けられています。ですから、Bキーだけを押すと、11111011＝FBHになるわけです。

さて、2つのキーを同時に押したらどうなるでしょうか。AキーとCキーを同時に押すと、表示はF5となります。

Aキーはビット1に、Cキーはビット3に位置付けられていますから、両方押すと11110101BでF5Hとなるわけです。

1つ、2つときましたから今度は3つです。3つになったからといって、基本的な考え方は同じです。Aキー、Bキー、Cキーを同時に押すと、ポート02Hの値はF1Hになります。ところが、です。IキーとJキーとBキーを同時に押して下さい、もちろん、IキーとJキーはポート03Hにつながっていますから、ポート03Hを読まなければ、押されたかどうかはわかりません。それはともかく、表示を見て下さい。F9になっています。ポート02Hのキーの中では、Bキーしか押していないのですからFBになるはずです。

F9ということは2進で11111001ですから、Aキーも押されたことになっているわけです。でも実際には押していません。これはなぜでしょうか。もう一度キーの配列を見てみましょう。

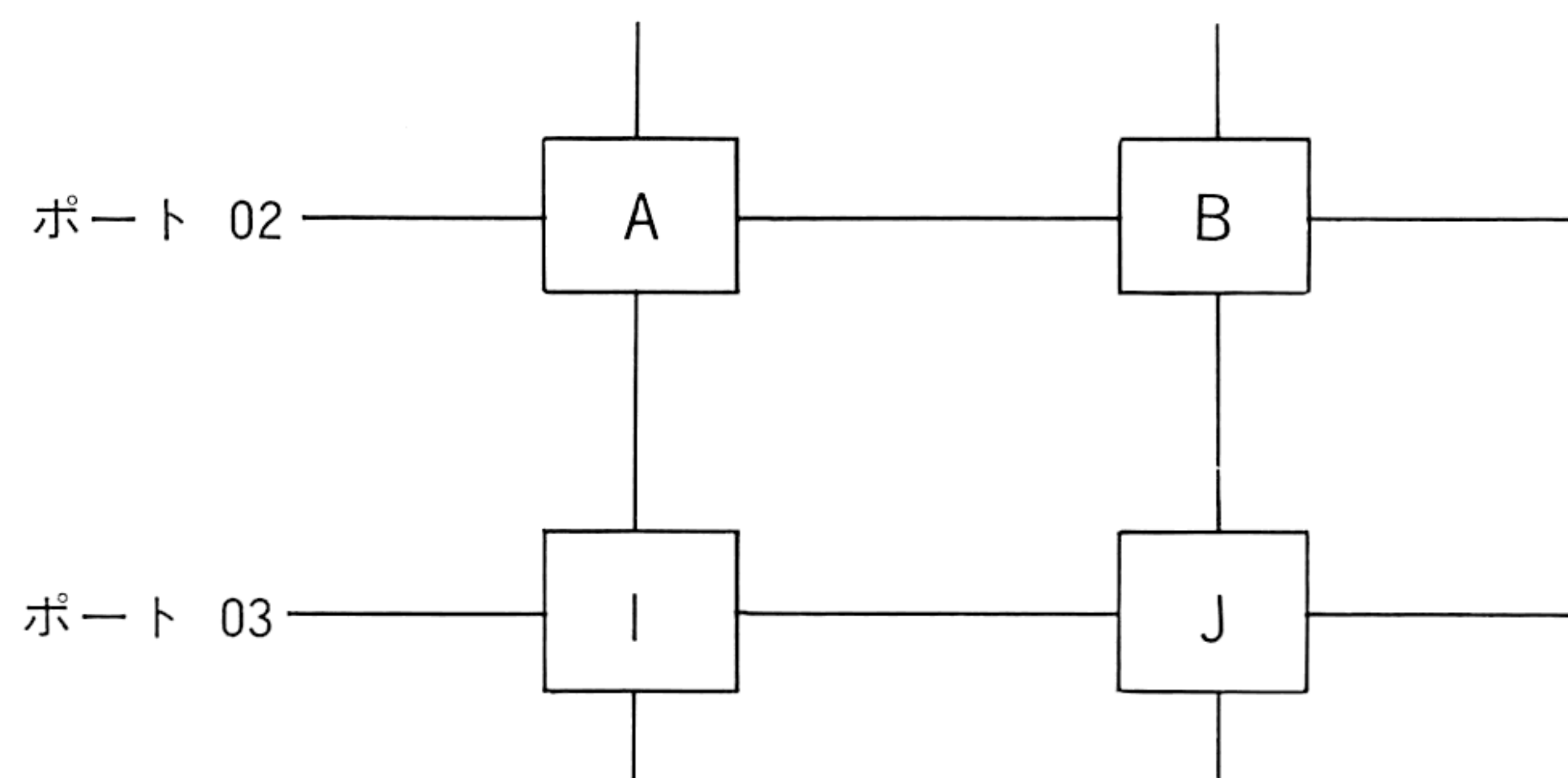



図5-1-B　キーの配列例

すると、[A]キーというのは[B]キー、[I]キー、[J]キーとともに四角形を形成しています。このような時は、4つのキーのうち3つを押すと、残りの1つも押されたことになってしまいます。別の四角形を形成しているキー、たとえば、[A], [Q], [U], [E]のキーでもためしてみて下さい。

これは、電気的にみると次のような理由によります。(この図の流れの向きはわかりやすくするために実際とは逆になっています。電子の流れと考えるとよいかもしれません。)

ビット1
ビット2
Aキー
Bキー
ポート 02
ポート 03
Iキー
Jキー

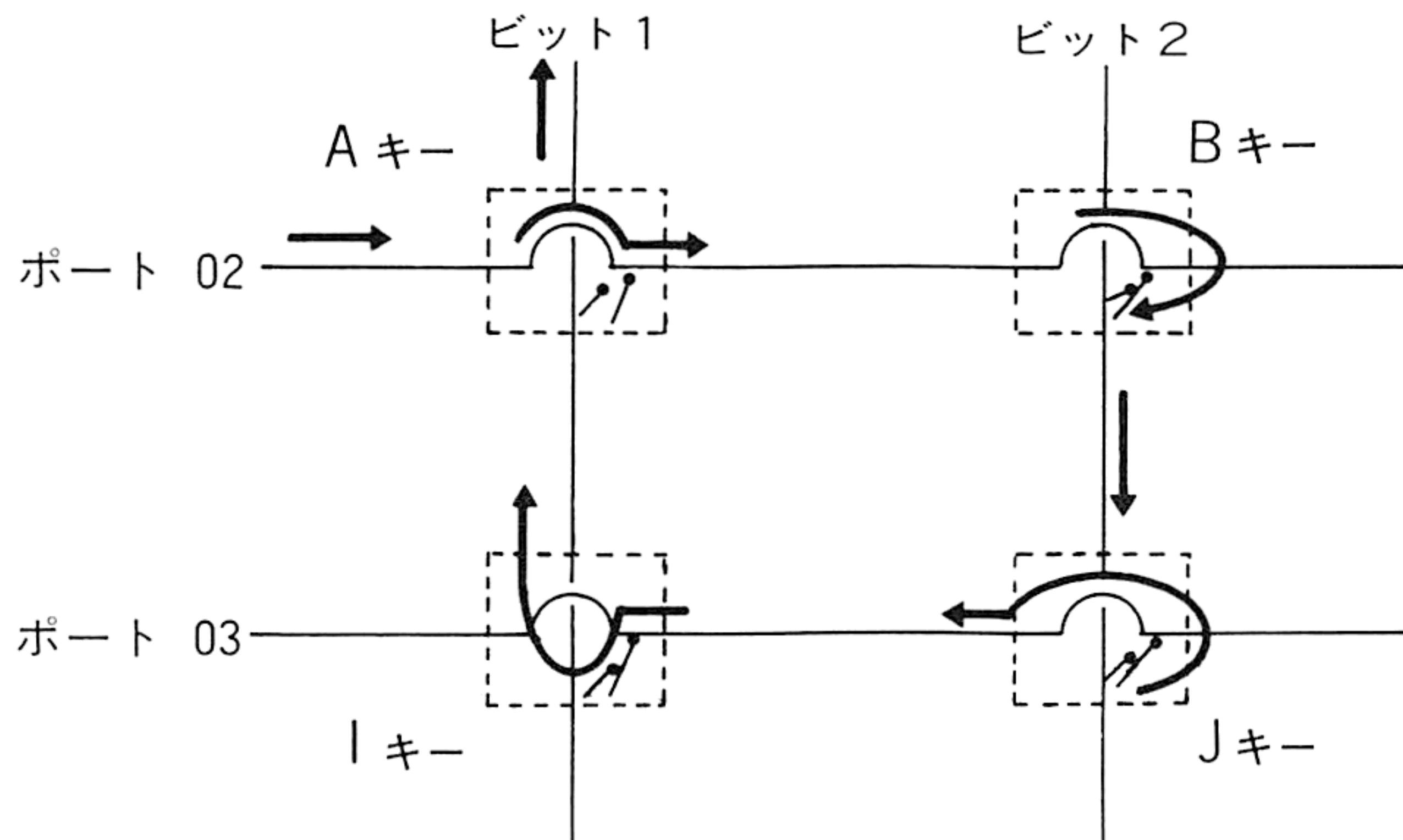



図5-1-C 電気の流れ

ポート02Hをたどっていった場合、[B], [J], [I]のキーが押されていると、[B]キーを通してビット2へ信号が行くのと同時に、[B]キーから[J]キー→[I]キーを通してビット1へも信号が行きます。したがって、あたかも[A]キーを押しているように見えるわけです。なお、[CTRL], [CAPS], [SHIFT], [カナ]、[GRPH]の5つのキーは、他のキーと同時に押すことを前提としていますので、ダイオードを使ってこのようなことが起こりにくいようにしてあります。また、押したままで使う[CAPS]キーと同じビット(ビット7)のキーにもダイオードが使ってあり、たとえば[CAPS]を押したままでMONと素早く打ち込んでも正常に入力されるようになっています。

このように、同時に3つ以上のキーを押すと、押していないキーが押されたかのような動きをすることがあるわけですが、この、ポートを読んで押されたキーを判断するという方法は、同時に複数のキーの状態を認識する方法として、ゲームなどでは、よく用いられています。

5-2　キースキャン

PC-8801mkⅡはソフトウェアによってキースキャンを行なっています。前述の方法によって、すべてのキーボード用のポートを読み、その値によってキーが押されたかどうかを判断しているわけです。

N88-BASICではキーの先行入力が可能です。これは割り込みを使用して実現しています。PC-8801mkⅡには、VRTC割り込みというものがあり（詳細は割り込みの章を見てください。）10数msごとに割り込みがかかっています。この割り込みルーチンでキーボードのスキャンをし、押されたキーを判別し、そのアスキーコードをキーバッファに入れます。このVRTC割り込みは、通常はいつでもかかるようになっていますから、いつキーを押してもそのキーは入力されます。

ところで、ワークエリアを操作すると、キースキャンを行なわなくさせることができます。これはプログラム中で、一時的にキー入力を受け付けたくない場合とか、キースキャンには結構時間がかかりますから、キー入力は必要ないが高速で計算したいというときに使えます。（数％速くなります）操作するワークエリアはE6CDHです。

POKE　&HE6CD，255……キースキャンを止める。

POKE　&HE6CD，0………キースキャンを再開する。

ここで注意しなければならないのは、キースキャンを止める必要がなくなったら、必ずキースキャンを再開しておくことです。これがないと、いっさいキー入力を受けつけなくなりますので、何もできません。もし、誤って再開命令を入れ忘れた時には、ストップリセット（STOPキーを押しながらリセットボタンを押す）を行なって下さい。

5-3　キーバッファ

キーボードがスキャンされ、押されたキーがあると、そのキーのデータ（アスキーコード）がこのキーバッファに入れられます。BASICのキー入力文はすべてこのバッファから文字をとり出します。

キー入力バッファは2-3で述べたようにキュー形式をとっています。キュー長は31文字で、このため31文字までの先行入力ができます。

キューアドレス：EFD9～EFF8H

キューテーブル

プットオフセット：EFCDH

ゲットオフセット：EFCEH

バックキャラクタ：FFCFH

キュー長　　　　：EFC0H

キューアドレス　：EFC1H，EFD2H

これを利用して、プログラム中で、ある文字列をあたかもキーボードから打ったような動作をさせることができます。

①文字列の長さが31文字以内のとき。

・キューの先頭(EFD9H番地)から文字列を書き込みます。

・プットオフセット(EFCDH番地)に文字列の長さ-1を入れます。

・ゲットオフセット(EFCEH番地)に1FHつまり31を入れます。

こうすると、キー入力待ちになった時に、その文字列をキーボードから打ち込んだのと似た動作をします。

実際にやってみましょう。

リスト　5-1

```
10 A$="TechKnow 8800mkII  SYSTEM SOFT"
20 FOR I=1 TO LEN(A$)
30   POKE &HEFD8+I,ASC(MID$(A$,I,1))
40 NEXT
50 POKE &HEFCD,LEN(A$)-1
60 POKE &HEFCE,31
Ok
run
Ok
TechKnow 8800mkII  SYSTEM SOFT
```

ところで、このようにソフト的にファンクションキーの動作をさせるのは、もっと簡単にできます。「5-6-4 ソフトファンクションキー」を見て下さい。

②文字列の長さが32文字以上のとき。

・メモリ上の適当な場所に文字列を書き込みます。

・プットオフセットに文字列の長さ-1を入れます。

・キュー長(EFC0H番地)に、2^n-1を満たし、文字列の長さよりも大きな値を入れます。例えば40字の文字列の場合は$2^6-1=63$です。

・キュー長と同じものをゲットオフセットにも入れます。

・キューアドレス(EFD1,2H番地)に文字列の先頭アドレスを入れます。

つまりキューの位置を移動するわけですが、動かしたままではいけませんので、あとで元に戻してやる必要があります。これは、35D9H番地をCALLすることによって簡単に行なうことができます。

キューの移動ということで面白いことをやってみましょう。

リスト　5-2

```
poke &Hefd1,&Hc8 : poke &Hefd2,&Hf3 : console 1,25
```

を実行します。キューを画面上に移したわけです。何かキーから入力すると、画面の左上に同じものが現われます。コントロールキーやカーソル移動キーを押すと対応するキャラクタが現われます。キューを元にもどすにはSTOPキーを押します。

5-4　キーバッファのクリア

N88-BASICはキーの先行入力が可能です。これはたいへん重宝な機能なのですが、反面困ることがあります。知らず知らずのうち⏎キーを押してしまって、あとであわてた経験はどなたでもおありでしょう。これがダイレクトモードであればSTOPキーを押せば良いのですが、プログラム実行中だとそういうわけにも行きません。あらかじめ適所適所でキーバッファをクリアしてやらなければなりません。

キーバッファのクリアをBASICで行なうとこうなります。

```
*KEY.CLEAR：　IF　INKEY$<>""　THEN *KEY.CLEAR
```

または、

```
WHILE　INKEY$<>""　：WEND
```

この機能をROM内のルーチンをCALLすることにより簡単に、かつ素早く行なうこともできます。アドレスは35D9Hですから

```
DEF USR=&H35D9：A=USR(0)
```

または

```
KEU.CLEAR=&H35D9：CALL KEY.CLEAR
```

とします。

5-5　キー入力ステートメント活用テクニック

5-5-1　INPUT文と疑問符

N-BASIC文では、入力待ちになる時、プロンプト文に続けて'?'が出力されますが、N88-BASICでは、出力させなくすることもできるようになっています。INPUT 文を入力する際、プロンプト文の後に"," を入れて下さい。

これは、プログラムを作る上でも便利なもので、たとえば、16進数を入力させる場合、次のようなプログラムにすることもできます。

リスト　5-3

```
10 INPUT "Start Address .. &H",SA$
20 S.ADRS=VAL("&H"+SA$)
Ok
run
Start Address .. &H■

               ↓

run
Start Address .. &H44a5
Ok
```

5-5-2　INPUT　WAIT文と待ち時間

INPUT WAIT文は待ち時間だけキーボードからの入力を待つINPUT文です。普通のIN-PUT文は、このWAITが無限大のものと思えば良いでしょう。

この待ち時間は、(INPUT WAITの後で指定した値)×0.1秒ということになっていますが、どのくらいの大きさまで使えるのでしょうか？実際にやってみると、32768以上でOverflowになってしまいますね。

つまり、待ち時間というのは、整数型の数値または変数でなければならないわけで、もしそれ以外(10.3など)であれば、整数型に変換され、それが待ち時間となります。

となると、最大待ち時間は3276.7秒ということになりますが、本当にそうでしょうか。実は、この値には、負の数も使えるのです。(整数型というのは-32768～32767というのを思い出して下さい)試しに、INPUT　WAIT　-1, Aとすると、6553.5秒間待つことになります。

なお、待ち時間を0(0.4などでも同様)にすると、Illegal　function　callとなりますので注意して下さい。

また、このINPUT　WAIT文は、キー入力がないということを前提とすると、一種のタイマーとして使うことができます。これは一秒ごとにカウントするプログラムです。

リスト　5-4

```
100 INPUT WAIT 10,"",
110 PRINT I,
120 I=I+1
130 GOTO 100
```

5-5-3　LINE INPUT文と数値の代入

INPUT文では、入力するデータの個数や型が違っていると"？Redo from Start"などというメッセージが出力され、時には画面のレイアウトがこわされたりすることがあります。これをなんとかしようということで登場して来るのが、このLINE INPUT文です。

次の例は、名前と年令を","で区切って入力させるサブルーチンですが、入力ミスがある場合は、画面をこわすことなく再入力させることができます。

リスト　5-5

```
100 *D.INPUT
110  LOCATE 0,10 : LINE INPUT "ﾅﾏｴ , ﾈﾝﾚｲ ? ",LN$
120  SP=INSTR(LN$,",") : IF SP<2 THEN *BAD
130  NM$=LEFT$(LN$,SP-1)
140  AGE=VAL(MID$(LN$,SP+1))
150  IF NM$<>"" AND AGE>0 THEN *OK
160 '
170 *BAD
180  LOCATE 0,10 : PRINT "... ﾏﾁｶﾞｲ ...";STRING$(9,9);
190  BEEP
200 GOTO *D.INPUT
210 '
220 *OK
230  PRINT "ﾅﾏｴ  ...  ";NM$
240  PRINT "ﾈﾝﾚｲ ...  ";AGE
250 END
```

INPUT NM$, AGEと書くよりもかなり複雑なものになっていますが、不特定多数の人に使われるプログラム(ビジネス・アプリケーション)などでは、これくらい注意を払ったプ

ログラムにしたいものです。

5-5-4　INP関数とWAIT文

INP関数はキー入力用のものではありませんが、5-1で述べた方法を用いることにより、入力ポートを通して、キーボードの状態を知ることができます。

INP関数は、機械語のIN命令と同じものです。

BASIC　　　　　　　　　アセンブラ

A=INP(PORT)　　＝　　IN　A,(PORT)

この関数を使うと、INKEY$などに比べて高速のキーセンスが可能ですし、2つ以上のキー状態がわかりますから、ゲームなどにはもってこいです。たとえば、カーソルキーを判別するプログラムは次のようになります。

リスト　5-6

```
100 '
110 '  Sence  Cursor keys
120 '
130 *LOOP
140  PT8=NOT INP(8)
150  PTA=NOT INP(10)
160   IF  PTA AND 2  THEN PRINT "DOWN"
170   IF  PTA AND 4  THEN PRINT "LEFT"
180   IF  PT8 AND 4  THEN PRINT "RIGHT"
190   IF  PT8 AND 2  THEN PRINT "UP"
200 GOTO *LOOP
```

WAIT文もポートの状態を見るものですから、キー入力に使えます。WAIT文はちょっとわかりにくいのですが、INP関数を使ってWAIT　PORT,VAL1,VAL2を書くと次のようになります。

WHILE((INP(PORT)XOR VAL2)　AND　VAL1)=0:WEND

ですからWAIT文を使ったものは必ずINP関数で書きなおすことができるわけで、逆にいうとWAIT文はあまり使う必要がないことになります。WAIT文の特長としては、高速で判別できる、入力待ちの時にキー割り込みがきかない(STOPできない)、ということがあります。

なお、INP,WAITを使った場合でも、押されたキーのデータは、キーバッファに入ってしまいます。これをクリアする方法は、5-4を見て下さい。

5-5-5　INKEY$でカーソル表示

INKEY$関数でキーセンスを行なう場合、通常はカーソルが表示されませんが、表示させた方がよい場合があります。その方法は次の通りです。(USR関数で行なってもかまいません。)

・カーソルON

CSRON=&H4290 : CALL CSRON

・カーソルOFF

CSROF=&H428B　:　CALL　CSRPF

それでは、これを使った例を見てみましょう。

リスト　5-7

```
100 '
110 ' INKEY$ with cursor
120 '
130 *LOOP
140 GOSUB *C.ON
150  A$=INKEY$ : IF A$="" THEN 150
160  PRINT A$,
170 GOTO *LOOP
180 END
190 '
200 ' Cursor ON
210 *C.ON
220  DEF USR=&H4290 : A=USR(0)
230 RETURN
240 ' Cursor OFF
250 *C.OFF
260  DEF USR=&H428B : A=USR(0)
270 RETURN
```

なおこの方法は、INP関数、WAIT文などでも用いることができます。

5-6　ファンクションキー

5-6-1 ファンクションキーデータの格納アドレス

ファンクションキーの内容はE6F2H～E791H番地に格納されています。

F.1	E6F2H～E701H	F.6	E742H～E751H
F.2	E702H～E711H	F.7	E752H～E761H
F.3	E712H～E721H	F.8	E762H～E771H
F.4	E722H～E731H	F.9	E772H～E781H
F.5	E732H～E741H	F.10	E782H～E791H

各キーにつき16バイトが割り当てられていますが、ターミネータとして00が必要ですので、定義できるのは15文字までとなっています。

ファンクションキーを実現するアルゴリズムは簡単で、ファンクションキーが押されると、定義されている文字列をキューにほうり込むだけです。もしキューが一杯になったら残りの文字は無視されます。

5-6-2　2つのファンクションキーをつなげる

さきほどターミネータとして00が必要だと述べましたが、これをなくしてしまうとどうなるでしょう。

リスト　5-8

```
key 1,"ABCDEFGHIJKLMNO"
Ok
key 2,"0123456789"
Ok
poke &He701,asc("@")
```

ここで[f·1]を押します。

リスト　5-9

```
ABCDEFGHIJKLMNO@0123456789
```

[f·1]と[f·2]がつながってしまいました。こうすることによって16文字以上の定義もできます。しかしキューの長さが31文字ですので、32文字以上の文字列を定義しても無意味です。

5-6-3　ファンクションキーの初期化

ファンクションキーの初期データはN88-BASIC ROM内の01B0H～024FHに入っています。電源ON(リセット)の時にこのデータがE6F2H～E791Hに転送されてくるわけです。従ってファンクションキーを初めの状態に戻すには、もう一度データを転送してやればよいわけです。これをBASICで行なうと次のようになります。

リスト　5-10

```
100 '
110 '    Function keys intialize sub.
120 '
130 *F.INIT
140  FOR I=0 TO 159
150    POKE &HE6F2+I,PEEK(&H1B0+I)
160  NEXT
170 RETURN
```

これはサブルーチンになっていて、必要な所にGOSUB　*F.INTを入れておくとファンクションキーが初期化されます。しかし表示は変わりませんので、シフトキーを押すか、console,, 1を実行して下さい。

次にこれを機械語で行なった例を示します。

リスト　5-11

```
100 '
110 ' Function Key INIT Command ( M.L. Version )
120 '
130 FOR I=&HF260 TO &HF270
140   READ D$ : POKE I,VAL("&H"+D$)
150 NEXT I
160 POKE &HEEA8,&H60 : POKE &HEEA9,&HF2
170 DATA E5,21,B0,01,11,F2,E6,01,A0,00,ED,B0,CD,79,3F,E1,C9
```

このルーチンを実行すると、新しいコマンド 'POLL' ができ、他のステートメントと同じように使えます。プログラム上の必要な箇所にPOLLと入れると、そこでファンクションキーが初期化されます。

5-6-4　ソフトファンクションキー

ソフトファンクションキーとはその名の通りソフトウェアでファンクションキーの役割をさせようというものです。その方法は次の通りです。

①メモリ中に文字列を用意します。文字列の最後は00Hで終わっていなければなりません。

②E6CDH番地に00Hでない値を書き込みます。

③F003, 4H番地に文字列の先頭アドレスを下位、上位の順に入れます。

④E5CEH番地に00Hでない値を書き込みます。

⑤F00CH番地にファンクションキー割込みを定義していないファンクションキー番号-1を書き込みます。

⑥E6CDH番地に00Hを書き込みます。

例をあげてみましょう。

リスト　5-12

```
10 A$="This is a pen."
20 P=VARPTR(A$) : KEY OFF
30 POKE &HE6CD,255
40 POKE &HF003,PEEK(P+1) : POKE &HF004,PEEK(P+2)
50 POKE &HE6CE,255
60 POKE &HF00C,0
70 POEK &HE6CD,0
Ok


run
Ok
This is a pen.
```

このうち、肝心なのは40行と50行です。30行はキー入力を止めるためで、70行は再開するためです。そうしないと実行中にキーを押すとうまく動かないことがあるからです。60行は、割込みを起こさないためです。ここに割込みが定義されていて、ON状態であるキー番号ー1をPOKEすると、そのキーの割込みが起こってしまい、ファンクションキーの働きをしません。しかしKEY　OFFがしてあれは60行はなくてもかまいません。

この方法を使うと、プログラムをPRINT文で書いた後、カーソルをその行にもって行き、⏎を押したことにすれば、自動的にプログラムを増やすことが出来ます。DATA文の自動作成などに有効です。

5-7　比較表

★文字列の入力方法の比較表

		INPUT	INPUT WAIT	LINE INPUT	LINE INPUT WAIT	INPUT $(X)	INKEY$
		文	文	文	文	関数	関数
入力表示	プロンプト文	可能	可能	可能	可能	不可能	不可能
	プロンプトマーク？	可能	可能	無	無	無	無
	カーソル表示	有	有	有	有	有	可能＊1
	エコーバック	有	有	有	有	無	無
	入力待ち	待つ	指定した時間だけ待つ	待つ	指定した時間だけ待つ	待つ	待たない
データ入力	、の入力	ダブルクォートで囲めば可	ダブルクォートで囲めば可	可能	可能	可能	可能
	" の入力	文字列の最初でなければ可	文字列の最初でなければ可	可能	可能	可能	可能
	コントロールコード カーソルキーの入力	不可	不可	不可	不可	可能	可能
	複数の変数への入力	可能	可能	不可	不可	不可	不可
	入力文字数	254文字以内	254文字以内	254文字以内	254文字以内	指定した文字数(255文字以内)	1 文字
	入力終了	⏎ キー	⏎ キー	⏎ キー	⏎ キー	自動	―――
	入力文字なしで⏎キー	ヌルストリング	ヌルストリング	ヌルストリング	ヌルストリング	CHR$ (13)	CHR$ (13)
STOPキー	BREAK表示	あり	あり	あり	あり	なし	あり＊2
	CONTによる再開	可能	可能	可能	可能	不可	可能＊2

＊1：通常はカーソルは表示されませんが、表示させせることも可能です。本文を見て下さい。
＊2：INKEY $がブレークされたりCONTされたりするわけではありません。
　　INKEY $自体はSTOPキーをCHR $(3)として受けつけることも可能です。

★キーセンス比較表

	INPUT$ (1)	INKEY$	INP (X)	WAIT
入力待ち	待つ	待たない	待たない	待つ
STOPキー	中断(CONT不可)	中断*	中断*	中断しない
複数キーの同時入力	不可	不可	可能	不可な場合もある
入力文字の判断	簡単	簡単	複雑	複雑
カーソル表示	表示する	表示させることも可	表示させることも可	表示させることも可
キー割り込み	きかない	きく*	きく*	きかない

＊入力待ちがないためINKEY $, INP(X)自体はSTOPキーやキー割り込みとは関係ありません。

第6章　テキスト画面

テキスト画面は下図のようにテキストVRAM、DMAコントローラ、CRTコントローラを通してCRTに表示されます。

CPU ←→ テキストVRAM → DMAコントローラ → CRTコントローラ ←→ CRT

（グラフィック画面）

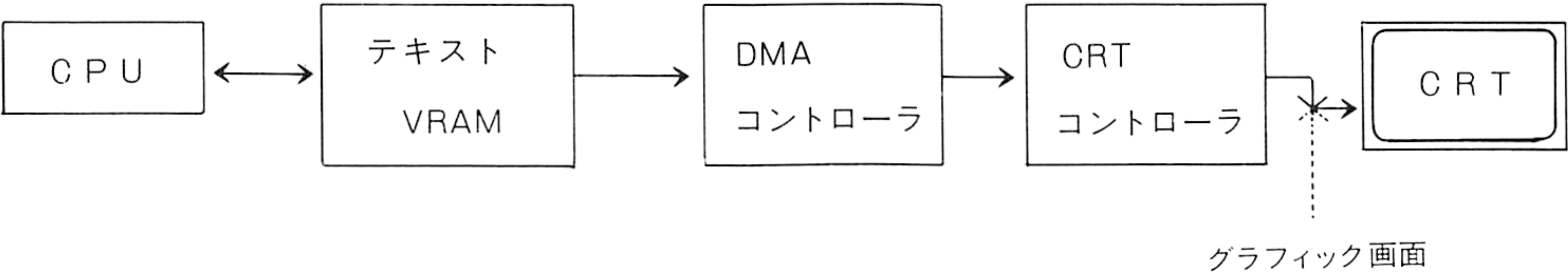



図6-A　テキスト画面の表示の流れ

6-1　テキストVRAM

6-1-1　テキストVRAMと画面

CRTの画面とテキストVRAMとの対応を図6-1-A、図6-1-Bに示します。テキストVRAMは、メインメモリ上のF3C8HからFF7FHまでの約3Kバイトに位置していて、表示エリアとアトリビュートエリアから成り立っています。表示エリアには画面に表示するデータが格納されていて、画面上の1文字と1対1に対応しています。

行数	0	1 …(桁数)… 79	80			
0	F3C8	F3C9………F416	F417	F418	F419………F43E	F43F
1	F440	F441………F48E	F48F	F490	F491………F4B6	F4B7
2	F4B8	F4B9………F506	F507	F508	F509………F52E	F52F
:	:	表示エリア	:	:	アトリビュートエリア	:
19	FCB0	FCB1………FCFE	FCFF	FD00	FD01………FD26	FD27
20	FD28	FD29………FD76	FD77	FD78	FD79………FD9E	FD9F
:	:		:	:		:
24	FF08	FF09………FF56	FF57	FF58	FF59………FF7E	FF7F

図6-1-A　80桁モードのVRAM

行数	0	1 …(桁数)… 38	39			
0	F3C8	F3CA………F414	F416	F418	F41A………F43E	F43F
1	F440	F442………F48C	F48E	F490	F491………F4B6	F4B7
2	F4B8	F4BA………F504	F506	F508	F509………F52E	F52F
:	:	表示エリア	:	:	アトリビュートエリア	:
19	FCB0	FCB2………FCFC	FCFE	FD00	FD01………FD26	FD27
20	FD28	FD2A………FD74	FD76	FD78	FD79………FD9E	FD9F
:	:					:
24	FF08	FF0A………FF54	FF56	FF58	FF59………FF7E	FF7F

図6-1-B　40桁モードのVRAM

40桁モードでは、表示エリアのRAMの偶数アドレスを使用します。また、20行モードでは、行数0～19までの位置にあるRAMを使用します。

アトリビュートエリアとは、表示画面の修飾(色、リバース、ブリンク等の指定)を行なうためのデータエリアで、2バイトで1つのアトリビュートを指定します。アトリビュートエリアは各行にあり、行あたり40バイトですから、1行で最大20個までアトリビュートの指定ができます。アトリビュートの使い方については後述します。

ヌルラインというのは画面がスクロールしたときに現われる新しい行をうめるデータです。たとえばPOKE　&HFF89,64としてからROLL UPキーを押すと、最下行の10桁目に@が現われます。CLRキーを押すともとに戻ります。

カーソル位置からVRAMのアドレスを求めるには次のようにします。

BASICによる方法

・80桁モード

```
DEF  FNVRAM(X,Y)=&HF3C8+120*Y+X
```

・40桁モード

```
DEF  FNVRAM(X,Y)=&HF3C8+120*Y+2*X
```

どちらもFNVRAM(桁，行)によって対応するアドレスが求められます。

機械語による方法

Hレジスタに(桁+1)、Lレジスタに(行+1)を入れて、429DH番地をコールすると、HLレジスタペアにVRAMのアドレスが得られます。

たとえば、5桁・20行の場合は、

```
LD      HL，1506H
CALL    429DH
```

とします。

6-1-2　テキストVRAMアドレスの移動

VRAMの位置は、N-BASICではF300H番地からに固定されていましたが、N88-BASICでは、ワークエリア中のポインタ(E6C4，C5H)で与えられます。つまり、この値を変えることにより、VRAMの位置を別のところへ移すことができるわけです。

それでは実際にやってみましょう。

まず3120バイト分のRAMエリアを確保します。(VRAMは3000バイトなのですが、ヌルライン用にVRAMの後に120バイト必要です。)

CLEAR, &HCF00

つぎにポインタを書き変えます。

POKE &HE6C5, &HCF：POKE &HE6C4, 0

これで移ったのですが、キーを押してもカーソルが動くだけで画面はもとのままですね。これは、DMAコントローラがまだ前のままの状態であるためです。これを再設定するには、WIDTH文を実行します。(一応CLRキーを押した後で行なって下さい。)

これで必要な操作は終わりです。

POKE &HCF00, ASC("A")

を実行すると、左上にAという文字が出るはずです。これでVRAMは移動し、CF00H番地からDAB7H番地を占めることになりました。F3C8H番地から後のもとのVRAMは使用されなくなっています。

WIDTH文ではなく、CMD TEXT ONまたは後で紹介するDMAC再設定プログラム(6.4 DMAコントローラ)を使うと、画面をクリアせずにDMACを再設定できます。これを利用すると複数のテキスト画面を持つことができます。次のプログラムは2枚のVRAMを使ったデモプログラムです。(拡張命令を使っています。)

リスト 6-1

```
100 CLEAR,&HCF00
110 WIDTH 80,25
120 ON STOP GOSUB *STOPP : STOP ON
130 '
140 FOR N=0 TO 26
150   IF N MOD 2=1 THEN POKE &HE6C5,&HCF : POKE &HE6C4,0
160   IF N MOD 2=0 THEN POKE &HE6C5,&HF3 : POKE &HE6C4,&HC8
170   CLS
180   FOR I=0 TO 20
190     LOCATE RND*70,RND*22
200     PRINT CHR$(N+&H41)
210   NEXT
220   CMD TEXT ON
230 NEXT N
240 '
250 *STOPP
260 POKE &HE6C5,&HF3 : POKE &HE6C4,&HC8 : WIDTH 80
270 STOP OFF
280 END
```

なお、この実験を行なった後は

POKE &HE6C5, &HF3 ： POKE &HE6C4, &HC8 ： WIDTH 80

を実行して、VRAMをもとの位置にもどしておいてください。

6-2　アトリビュートエリア

PC-8801mkⅡではテキスト画面の制御にPC-8001(mkⅡ)と同じμPD-3301を使っています。これは、カラーやリバースなどの指定にアトリビュート(属性)方式を用いているものです。この属性は、前に示したようにVRAM上にアトリビュートエリアとして置かれ、N₈₈-BASICでは次のような構成になっています。N-BASICとN₈₈-BASICでは多少違いがありますが、本質的には同じものです。

アトリビュートエリアの先頭
1つの属性は，2バイトで表現されている

N₈₈-BASIC
(1行目)

F418H	F419H		F43EH	F43FH
桁数	属性		桁数	属性

属性コード

この桁までを次の1バイトの属性
コードに従った表示を行う

1行につき40バイト(20組)確
保されている
従って，属性の変化は，20回
まで許される

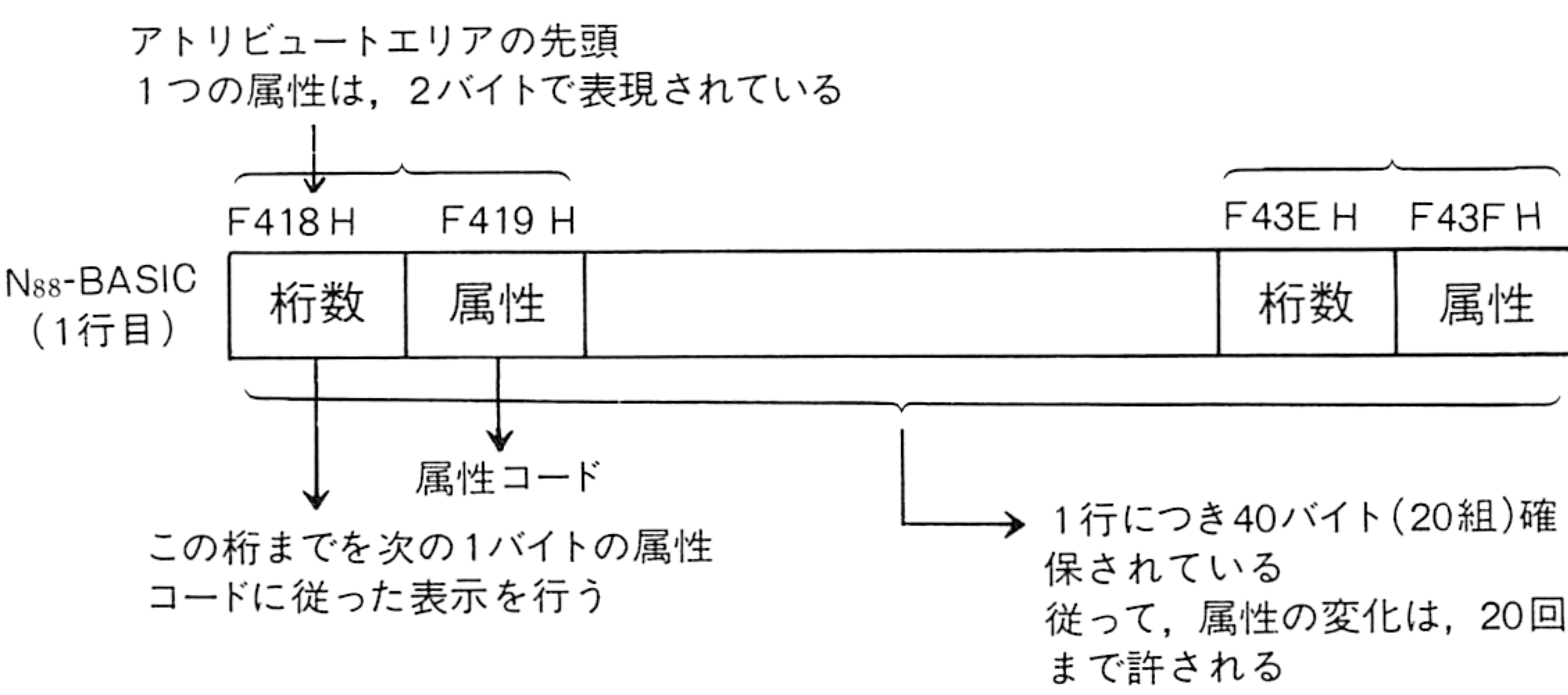



図6-2-A　アトリビュートエリア(N₈₈-BASIC)

アトリビュートエリアの先頭
1つの属性は，2バイトで表現されている

N-BASIC
(1行目)

F350H	F351H	F352H	F353H		F376H	F377H
桁数	属性	桁数	属性		桁数	属性

属性コード

この桁から次の1バイトの属性
コードに従って表示を行う

1行につき40バイト(20組)確
保されている
従って，属性の変化は，20回
まで許される

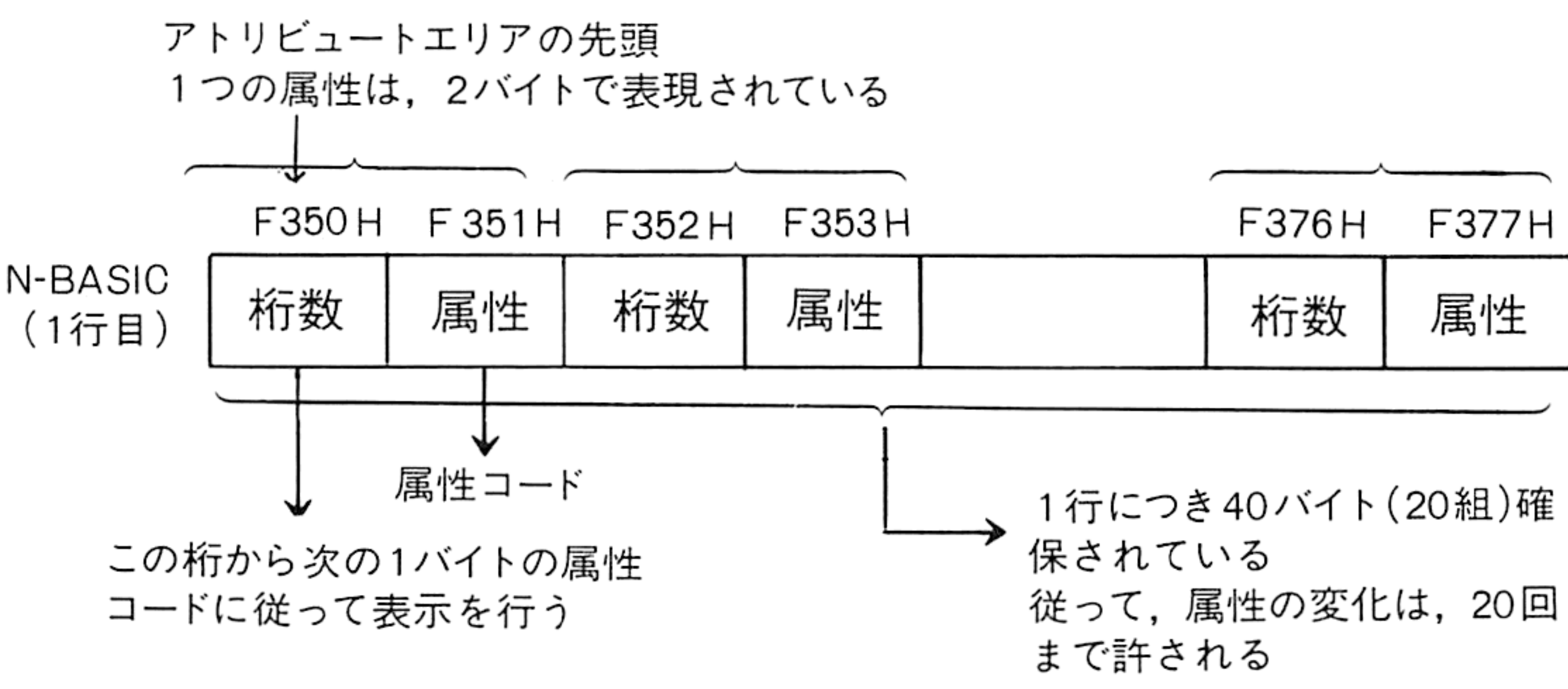



図6-2-B　アトリビュートエリア(N-BASIC)

6-2-1　属性コード

属性コードは1バイトよりなり、白黒モード、カラーモードのそれぞれの場合について次のような意味をもちます。

(a)白黒モード（CONSOLE , , ,0)

7	6	5	4	3	2	1	0
Graphic=1 Character=0	0	アンダーライン	アッパーライン	0	リバース	ブリンク	シークレット

000	ノーマル	COLOR 0
001	シークレット	COLOR 1
010	ブリンク	COLOR 2
011	シークレット	COLOR 3
100	リバース	COLOR 4
101	リバースシークレット	COLOR 5
110	リバースブリンク	COLOR 6
111	リバースシークレット	COLOR 7

(b)カラーモード（CONSOLE , , ,1)

7	6	5	4	3	2	1	0
0	0	アンダーライン	アッパーライン	0	リバース	ブリンク	シークレット

※カラーの時のLINE文に対応（N-BASIC)

このビットが0か1か2通りある。

7	6	5	4	3	2	1	0
緑	赤	青	Graphic=1 Character=0	1	0	0	0

000	黒	COLOR0
001	青	COLOR1
010	赤	COLOR2
011	紫(マゼンダ)	COLOR3

100	緑	COLOR4
101	水色(シアン)	COLOR5
110	黄	COLOR6
111	白	COLOR7

図6-2-C　属性コード

6-2-2　低解像度グラフィック

属性コードの図表にGraphicというのがありますね。これはN-BASICで使用している低解像度のグラフィックのことです。ハードウェアは同じなのですからN88-BASICモードでも使用できます。実際にやってみましょう。

リスト　6-2

```
10 CONSOLE 0,25,0,0 : WIDTH 80,25
20 FOR AD=0 TO 79
30    POKE &HFDA0+AD,AD
40 NEXT
```

表示エリア(この例ではFDA0H～FDEFH番地)のデータを上のように書き換えますと、画面の下のほうに

S_H　S_X　E_X　E_T　E_Q...J　K　L　M　N　O

と表示されます。ここでアトリビュートエリアのFDF0H, FDF1Hを50, 80に書き換えると、いままで表示されていた文字が

00 01 02 03 04 4B 4C 4D 4E 4F

アドレス FDAD FDA1 FDA2 …… FDED FDEE FDEF

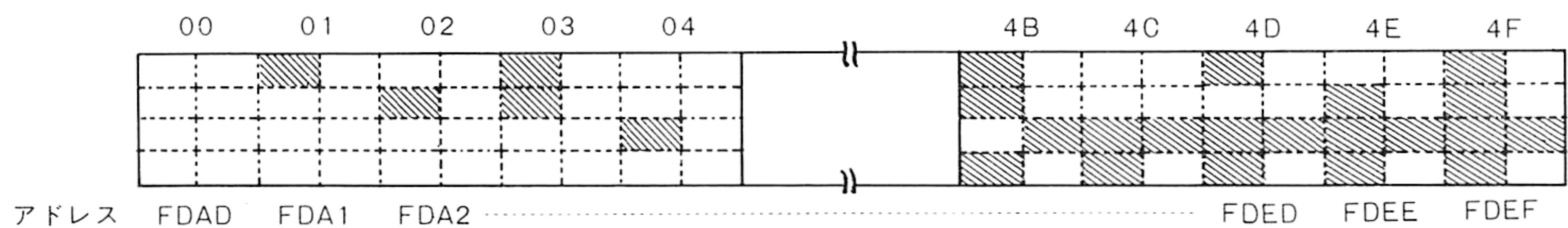



図6-2-D　低解像度グラフィック例

上のような模様に変わります。表示エリアのデータとこのグラフィックパターンの関係は次のようになっています。例として、データが34Hの場合をみてみます。

ビット0 ビット4
ビット1 ビット5
ビット2 ビット6
ビット3 ビット7

7 6 5 4 3 2 1 0
34H 0 0 1 1 0 1 0 0

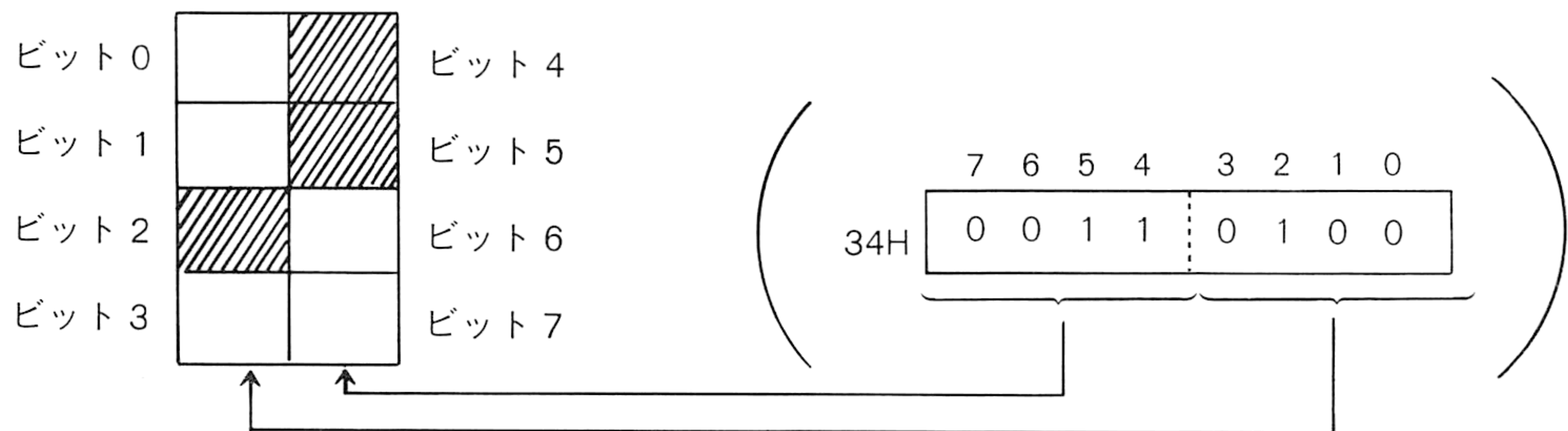



図6-2-E　低解像度グラフィックのマッピング

6-2-3　アトリビュート・セット

アトリビュートのセットの中でも、文字の色付けはBASICインタプリタで行なってくれます。ところがアンダーライン、アッパーライン、低解像度グラフィックモードなどについてはハードウェアには表示する能力があるのに、BASICではサポートされていないので、自力でやるしかありません。

そこでここでは、2つの方法でアトリビュートのセットを行なった例を示します。

①ワークエリアを操作する方法

PRINT文で文字を出力するときには、E6B4H番地に格納されている値でアトリビュートをセットします。E6B4H番地はCOLOR文でセットされるのですが、直接ここに属性コードを書き込むと、その値を使ってプリントされるようになります。たとえばここに20Hを入れると、以後の文字にはアンダーラインが付くようになります。ただし、COLOR文を実行すると値が再設定されてしまいますので、アンダーラインは付かなくなります。

白黒モードのときはこの方法でCOLOR文と同じような感覚でアンダーラインなどが付けられますが、カラーモードのときはちょっと問題が起こります。「6-2-1属性コード」のところを御覧になればわかるとおり、カラーモードでは属性コードが2種類あります。

N88-BASICでは色指定がある方の属性コードを使っています。こちらの属性コードを書き込む分にはよいのですが、もうひとつの機能指定がある方の属性コードを書き込むと、その属性で書いた文字からその行の最後まで、指定した機能が効いてしまいます。たとえば5文字目にアンダーラインをつけた文字を書くと、5文字目から行の最後までアンダーラインがついてしまうわけです。

この対策としては、機能をつけたくないところの最初にノーマルの属性で(POKE &HE6B4,0)文字を書く方法があります。

次の例は白黒モードで、入力した文字列にアンダーラインを引くプログラムです。

リスト　6-3

```
100 CONSOLE ,,,0
110 LINE INPUT "Input String:"; STRNG$
120 PRINT
130 '
140 POKE &HE6B4,&H20
150 PRINT STRNG$
160 COLOR 0
```

②ROM内ルーチンの利用

いまのはPRINT文が使うワークエリアを操作する方法でしたが、こんどはアトリビュートエリアに属性をセットする方法を紹介します。これはROM内のルーチンを利用すると簡単です。使い方は、HLレジスタに画面上のアドレス、Cレジスタに属性コードを入れて4351H番地をCALLするだけです。この方法は機械語でアトリビュートをセットするときにも使えます。

次の例はアトリビュート・コードを低解像度グラフィックにするものです。乱数で発生させた位置(横0～159, 縦0～99)に低解像度でPSETします。

リスト　6-4

```
100 '---- Write machine code
110  FOR I=&HF320 TO &HF329
120    READ DA$ : POKE I,VAL("&H"+DA$)
130  NEXT
140  DEF USR=&HF320
150 '
160  DATA 0E,00,7E,23,66,6F,CD,51,43,C9
170 '
180 '--- Sample program
190 CONSOLE 0,25,,1 : WIDTH 80,25
200 DEFINT A-Z
210 '
220 FOR I=0 TO 29
230    CLR=INT(RND*7)+1
240    X=INT(RND*80)
250    Y=INT(RND*20)
260    V.ADRS=&HF3C8+Y*120+X
270    LOCATE X,Y : PRINT CHR$(1);
280    GOSUB *ATR.SET.SUB
290 NEXT
300 '
310 END
320 '
330 *ATR.SET.SUB
340  POKE &HF321,CLR*&H20+&H18
350  DUMMY=USR(V.ADRS)
360 RETURN
```

6-3　CRTコントローラ

CRTC(μPD3301)は、DMACとともにテキスト用VRAMの表示に使用されています。

6-3-1　コマンド

CRTCには9つの命令がありますが、PC-8801mkⅡではそのうちの7つを使用できます。

コマンド0：イニシャライズ

画面パラメータの設定をします。WIDTH文・CONSOLE文では、このコマンドを使っています。なおこのコマンドで3301を再設定するときにはDMACも設定しなおさなくてはなりません。

命　　令	データ（パラメータ）							
	ビット　7	6	5	4	3	2	1	0
①　OUT　51H	0	0	0	0	0	0	0	0
②　OUT　50H	C／B	H 6	H 5	H 4	H 3	H 2	H 1	H 0
③　OUT　50H	B 1	B 0	L 5	L 4	L 3	L 2	L 1	L 0
④　OUT　50H	S	C 1	C 0	R 4	R 3	R 2	R 1	R 0
⑤　OUT　50H	V 2	V 1	V 0	Z 4	Z 3	Z 2	Z 1	Z 0
⑥　OUT　50H	A T 1	A T 0	S C	A 4	A 3	A 2	A 1	A 0

まず下の説明を見て、各ビットを1にするか0にするかを決め、16進に変換し、①～⑥を、この順番にOUTします。

①リセット

　DMA要求が停止され、スクリーンフォーマット1～5を書き込み可能にします。

②スクリーンフォーマット1

・DMA動作指定(C／B)

　0　：　DMAバーストモード

　1　：　DMAキャラクタモード

・1行当たりの文字数

H 6	H 5	H 4	H 3	H 2	H 1	H 0	1行当たりの文字数
0	0	0	0	0	0	0	2
0	0	0	0	0	0	1	3
0	0	0	0	0	1	0	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0	0	1	1	1	0	80

(1001110をこえる値は指定禁止)

③スクリーンフォーマット2

・ブリンキング時間指定(ブリンクの周期)

B 1	B 0	カーソルブリンク(画面)	アトリビュートブリンク(画面)
0	0	16	32
0	1	32	64
1	0	48	96
1	1	64	128

カーソルブリンクの明暗比率は1対1、アトリビュートブリンクは3対1です。つまり、B1　B0　=00としたときカーソルの明暗は8画面ごとに変わり、アトリビュートによるブリンクは24画面の間は明で、8画面の間は暗となります。

・1画面当たりの行数

L 5	L 4	L 3	L 2	L 1	L 0	1画面当たりの行数
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	64

④スクリーンフォーマット3

1行おきの表示指定(S)

0　：　通常指定

1　：　1行おき表示

行数指定において25行を指定し、S＝1とすると、1行目から13行目までの文字データが1行おきに画面に表示されます。

・カーソル表示モード

C 1	C 0	カーソル表示モード
0	0	ブリンクしないアンダーラインカーソル
0	1	ブリンクするアンダーラインカーソル
1	0	ブリンクしない反転ブロックカーソル
1	1	ブリンクする反転ブロックカーソル

1文字当たりのライン数を13以下の値に設定した場合、アンダーラインカーソル表示モードを指定しても表示はしません。

・1文字当たりのライン数

R4	R3	R2	R1	R0	1文字当たりのライン数
0	0	0	0	0	指定禁止
0	0	0	0	1	指定禁止
0	0	0	1	0	3
0	0	0	1	1	4
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	32

⑤スクリーンフォーマット4

・垂直帰線幅

V2	V1	V0	行　数
0	0	0	1
0	0	1	2
0	1	0	3
⋮	⋮	⋮	⋮
1	1	1	8

・水平帰線幅

Z4	Z3	Z2	Z1	Z0	文字数
0	0	0	0	0	指定禁止
0	0	0	0	1	指定禁止
0	0	0	1	0	指定禁止
0	0	0	1	1	指定禁止
0	0	1	0	0	6
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	33

⑥スクリーンフォーマット５

・アトリビュートおよび特殊制御文字モード指定

ＡＴ１	ＡＴ０	ＳＣ	アトリビュート	特殊制御文字
0	0	1	アトリビュート無	無効
1	0	1	ノントランスペアレント白黒	無効
1	0	0	ノントランスペアレント白黒	有効
0	0	0	トランスペアレント白黒	有効
0	1	0	トランスペアレントカラー	有効
0	1	1	指定禁止	
1	1	0	指定禁止	
1	1	1	指定禁止	

特殊制御文字を有効にした場合、特殊制御文字用として各行について２バイト余分に確保する必要があります。

・１行当たりの最大アトリビュート数

Ａ４	Ａ３	Ａ２	Ａ１	Ａ０	１行当たりの最大アトリビュート数
0	0	0	0	0	1
0	0	0	0	1	2
0	0	0	1	0	3
⋮	⋮	⋮	⋮	⋮	⋮
1	0	0	1	1	20

10011を越える値は指定禁止、また、最大アトリビュート数には特殊制御文字数も含まれています。

＜標準的なパラメータ＞

PC-8801mkⅡには、ハイスキャンCRTとノーマルCRTがあり、これによってCRTのイニシャライズパラメータは異なります。また、20行あるいは25行の表示数によっても異なります。

番　号	データ（16進）			
	ノーマルCRT		ハイスキャンCRT	
	20行	25行	20行	25行
②	CE	CE	CE	CE
③	93	98	93	98
④	69	67	73	6F
⑤	BE	DE	38	58
⑥	カラーモード：53		白黒モード：13	

コマンド１：表示の停止

DMA要求が停止されます。動作としては、スクリーンフォーマットを書き込まないイニシャライズコマンド(リセットコマンド)と同じです。

OUT　&H51, 0

コマンド2：表示の開始

ステータスをクリアし、DMA要求可の状態にして画面表示を開始します。

OUT　&H51, &H20　　通常表示

OUT　&H51, &H21　　反転表示

コマンド３：インタラプトマスク

割り込み要求にマスクをかけるかどうかを指定します。DMACをオートロードモードで使用しますので、インタラプト無効の状態にします。

OUT　&H51, &H43

コマンド４：ライトペン位置の読み出し

ライトペンが押されたときの行、桁座標を読み出します。この命令実行後、ライトペン検出ステータス(LP)はクリアされます。

OUT　&H51, &H60　　コマンドの発行

X　=　INP(&H50)　　桁位置の読み取り

Y　=　INP(&H50)　　行位置の読み取り

ライトペン信号が帰線中に入力された場合、Xの最上位ビットが１となります。

コマンド５：カーソルのON／OFF、位置の設定

カーソルのON／OFFと位置を指定します。

・カーソルOFF

```
OUT &H51, &H70
OUT &H50, 桁位置
OUT &H50, 行位置
```

・カーソルON

```
OUT &H51, &H71
OUT &H50, 桁位置
OUT &H50, 行位置
```

コマンド8：ステータスの読み出し

ステータスフラグを読み出します。

ポート 51H (IN)

ビット	7	6	5	4	3	2	1	0
	0	0	0	VE	U	N	E	LP

- LP：ライトペン検出
- E：表示終了割り込み発生
- N：特殊制御文字割り込み発生
- U：DMA アンダラン発生
- VE：画面表示有効

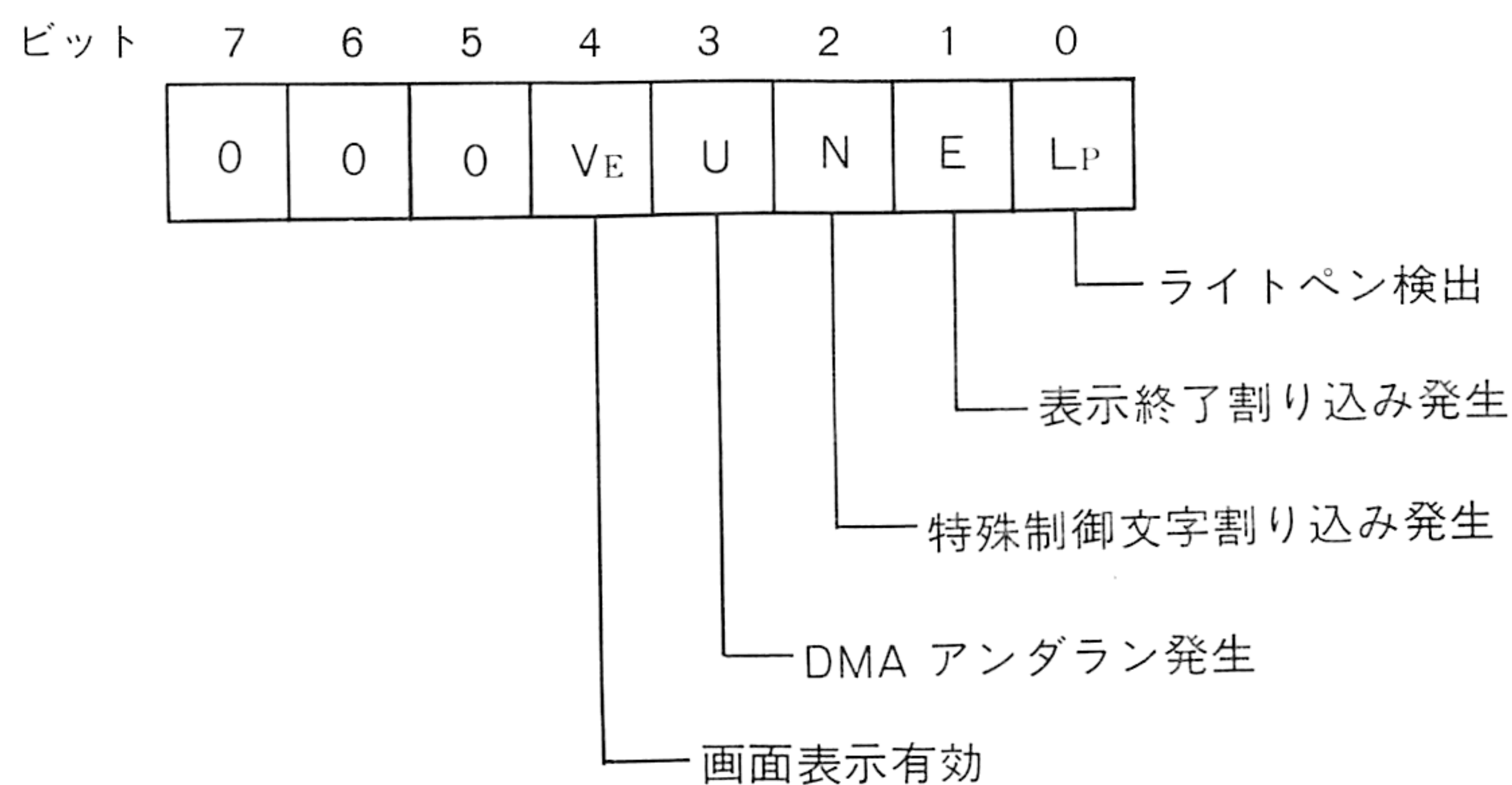



図6-3-A　CRTCのステータス

6-3-2　CRTC設定例

CRTCを設定するプログラム例を紹介します。これはCRTC, DMACを設定して、80桁／40桁、25行／20行、白黒／カラーを切り換えるプログラムです。ただし、VRAMのアトリビュートは変更しませんし、N88-BASICの画面モード用ワークエリアも変更しませんから、現在のモードと異なるモードを設定した場合には正常に表示されなくなります。また画面をクリアしません。

このプログラムはサブルーチンになっています。XWID, YWIDにそれぞれ横の桁数、縦の行数を入れ、COLRに白黒モードにしたいとき0、カラーモードにしたいとき1を入れてからGOSUB＊CRTSETとします。

リスト　6-5

```
100 '======== CRT SET SUBROUTINE ========
110 'entry:
120 '   XWID = 80 or 40
130 '   YWID = 25 or 20
140 '   COLR = 1 or 0
150 *CRTSET
160  IF XWID>60 THEN XWID=80 ELSE XWID=40
170  IF YWID>22 THEN YWID=25 ELSE YWID=20
180  OUT &HE4,0
190  IF YWID=20 THEN RESTORE *L20 ELSE RESTORE *L25
200  '
210  READ DT
220  DT = PEEK(&HE6C2) AND &HDF OR DT
230  POKE &HE6C2, DT : OUT &H31, DT
240  '
250  '---- 80 or 40, COLOR or MONO
260  IF XWID=80 THEN DT=PEEK(&HE6C0) OR 1 ELSE DT=PEEK(&HE6C0) AND &HFE
270  IF COLR THEN DT=DT AND &HFD ELSE DT=DT OR 2
280  POKE &HE6C0,DT : OUT &H30, DT
290  '
300  '---- Synchro off
310  DT = PEEK(&HE6C1) OR 8 : POKE &HE6C1,DT : OUT &H40,DT
320  '
330  '---- Set DMAC & CRTC
340  OUT &H51, 0
350  OUT &H68, &H80
360  OUT &H64, PEEK(&HE6C4)
370  OUT &H64, PEEK(&HE6C5)
380  READ DT : OUT &H65, DT
390  READ DT : OUT &H65, DT
400  IF INP(&H40) AND 2 THEN READ DT,DT,DT,DT: ' skip 80CRT
410  FOR I=0 TO 3 : READ DT : OUT &H50,DT : NEXT
420  OUT &H50, PEEK(&HE6B9) AND &H40 OR &H13
430  '
440  OUT &H51, &H43
450  OUT &H68, &HC4
460  OUT &H51, &H20
470  '
480  '---- Synchro on
490  DT = PEEK(&HE6C1) AND &HF7 : POKE &HE6C1, DT : OUT &H40, DT
500  '
510  OUT &HE4,3
520 RETURN
530 '
540 '---- CRTC & DMAC parameters
550 *L20 '  20 lines
560 DATA &h00,&h5F,&h89
570   DATA &hCE,&h93,&h73,&h38     :' 88 CRT (#2-#5)
580   DATA &hCE,&h93,&h69,&hBE     :' 80 CRT (#2-#5)
590
600 *L25 '  25 lines
610 DATA &h20,&hB7,&h8B
620   DATA &hCE,&h98,&h6F,&h58     :' 88 CRT (#2-#5)
630   DATA &hCE,&h98,&h67,&hDE     :' 80 CRT (#2-#5)
```

CRTCに設定するパラメータは最後のDATA文で指定されます。ここを変更すると標準的でないパラメータを設定することができます。たとえば590, 580, 620, 630各行の3番目のデータ(&H98または&H93)の9を1に変えると、カーソルのブリンク周期が短くなります。ほかのパラメータも色々変えてみるとおもしろいのですが、PC-8801mkIIというハードウェアの制限上、μPD3301のすべての機能を発揮することができず、変更できるパラメータも限られています。

μPD3301のアトリビュートには、N88-BASICで使っているトランスペアレントアトリビュートの他に、ノントランスペアレントアトリビュートがあります。これはアトリビュー

トエリアがなく、表示エリアと同じ領域をアトリビュート指定用に使用するものです。アトリビュートは1バイトで指定し、アトリビュートか通常の文字かはデータのMSBで判断します。

アトリビュートは表示エリアにありますので、アトリビュートが検出されますと、その部分だけ表示ができません。従って、ノントランスペアレントモードと呼ばれます。このモードを使うとアトリビュートの変更が一行20回までという制限がなくなるという利点がありますが、アトリビュートの部分が1文字分スペースになる(これは40桁モードにすればもともと奇数番目のデータは表示されませんから解決できます)、文字コード80H以上が表示できない(アトリビュートコードとして使用される)、カラーモードがないという欠点があります。

しかもN_{88}-BASICではまったくサポートされませんから、文字一つ表示するにも自分でVRAMをアクセスしなければならないなど、あまり実用的なモードではありません。

6-4 DMAコントローラ

DMAC(μPD8257)は、画面表示用にはテキストVRAM上のデータをCPUを介さずにCRTCへ送る役目をしていて、チャネル2を用いています。DMACには全部で4つのチャネル(0～3)があり、チャネル0、1はDMAタイプのフロッピィディスク用にバススロットを用いて使用できますが、チャネル3は使用できません。

各チャネルにはDMAの対象となるメモリの開始アドレスと、長さを設定する2つのレジスタ(各々16ビットと15ビット長)があります。ここにデータを書き込んだ後、全体を制御するモードセットレジスタを設定すると、DMA要求があったときに(テキスト画面表示ではCRTCが送る)DMAが起こります。実例としては次の節のDMAC再設定プログラムをみてください。

なお、各レジスタのI/Oアドレスは第16章のI/Oポート一覧表にあります。

6-4-1 DMAを止めて実行速度アップ

DMAが行なわれているときは、CPUは止まっています。したがってプログラムの実行速度は低下するわけです。長時間の計算やテキスト画面を使わないゲームなどではDMAを止めることによって、実行速度を30%近く上げることができます。DMAのオン・オフは拡張命令パッケージを使用しているときは簡単で、

```
DMAを止める：     CMD  TEXT  OFF
DMAを再開する：   CMD  TEXT  ON
```

とするだけです。

拡張命令を使用しないときは

```
OUT 81, 0  または  OUT 104, 0
```

によってDMAを止めることができます。ただし、BASICのコマンドレベルに戻ってもテキスト画面表示は回復しません。テキスト画面を再び表示させるにはWIDTH文を使うとよいのですが、画面がクリアされてしまいます。これを防ぐためには自分でDMACを再設定しなければなりません。そのためのプログラムは次のようになります。

```
10 OUT &H68, &H80                              :'一旦DMAを止める
20 OUT &H64, PEEK(&HE6C4)                      :'VRAMアドレス下位
30 OUT &H64, PEEK(&HE6C5)                      :'VRAMアドレス上位
40 IF PEEK(&HEF88)=25 THEN OUT &H65, &HB7:OUT &H65,
&H8B                                           :'25行のとき
50 IF PEEK(&HEF88)=20 THEN OUT &H65, &H5F:OUT &H65,
&H89                                           :'20行のとき
60 OUT &H68, &HC4                              :'DMACモード設定
70 OUT &H51, &H20                              :'CRTCに表示再開を指令
```

40, 50行は転送する長さ(15ビット)を下位、上位の順でOUTします。残りの最上位ビットは転送の方向を示すフラグで、1のときメモリから読み出すことを示します。

6-5 WIDTH文

6-5-1 WIDTH文のパラメータの省略

WIDTH文は、桁数と行数の2つのパラメータを持ちます。N-BASICでのWIDTH文はいずれも省略できますが(例えばWIDTH, 25、WIDTH, など)、N88-BASICでは行数の省略しかできません。したがって桁数がわかっていないときに、行数だけを変えたい場合は次のようにします。

```
WIDTH PEEK(&HEF89), 25(または20)
```

このEF89Hというのは画面の行数が入っているアドレスです。ちなみに行数はその前のEF88H番地に入っています。

ただしこれはBASICインタプリタが画面モードの値を参照するためのものであって、これらのアドレスに値をPOKEしたからといって、画面モードは変化しません。(他にCRTCのモード設定が必要です。)

6-5-2 WIDTH文とスクロールウィンドウ

N88-BASICにおけるWIDTH文は、N-BASICのものといくらか相違点があります。前に述べたパラメータの省略の他に、次のようなところが異なっています。

N88-BASIC	N-BASIC
WIDTH文を実行すると必ず画面がクリアされる	桁数を変化させた場合にのみ画面がクリアされる
WIDTH文の実行によりスクロールウィンドウが初期化される(CONSOL0.25が行なわれる)	スクロールウィンドウは変化しない

第１の点については困った点も出てきます。N-BASICでは、DMACやCRTCのモードの再設定(DMAをOFFにした場合のリカバー、OUT　81，33による画面反転をもとに戻すなど)のために、WIDTH，を実行すればよかったわけですが、N88-BASICではそれを行なうと、消えては困る画面が消されてしまうことになります。これは拡張命令のCMD TEXT ONや前出のDMAC再設定プログラムを使うと解決できます。

第２の点については、WIDTH文を実行する前にスクロールウィンドウを覚えておいて、実行後に再びCONSOLE文を実行すればよいわけです。

プログラム例を示しておきます。

リスト　6-6

```
100 STLN=PEEK(&HE6B2)-1
110 SCLN=PEEK(&HE6B3)-STLN
120 WIDTH 80,25
130 CONSOLE STLN,SCLN
140 PRINT CHR$(11);
```

6-6　PRINT文テクニック

6-6-1　PRINT文と改行

リセット直後と、WIDTH文を実行した後で、次の文を実行してみて下さい。

PRINT　STRING$(35,"＃");　"0123456789"

①リセット直後(40桁モード)

リスト　6-7

```
print string$(35,"#");"0123456789"
###################################01234
56789
```

②WIDTH40を実行後

リスト　6-8

```
print string$(35,"#");"0123456789"
###################################
0123456789
```

マニュアルの通り※なら②のようになるはずですが、リセットした後、WIDTH文を実行しないと、①のように改行されずに続けて表示されることになります。

(※PRINT文で表示する文字列の長さ、数値の長さが、現在のカーソルのある位置より後方にとれない場合、改行して表示される。)

これはワークエリアE64FH番地の値によるもので、一種のフラグとして考えてもかまいません。

リセットの時、ここにはFFHが書き込まれます。(これは②のような機能を無視するということです) また、WIDTH文を実行すると、その桁数が書き込まれます。つまり文字列等

を表示する場合、カーソルの位置がその値を超えるようであれば改行して表示するということを示しています。

ですから、N-BASICのように、①のような表示をしたければ、WIDTH文を実行した後であっても、E64FH番地にFFHをPOKEしてやればよいわけです。

これを使って次のようなこともできます。このプログラムは、80桁モードで平方根の値を表示するのに、画面の左半分だけに出力するものです。あたかも、WIDTH　PRINT（？）を実行したかのように動作していますね。

リスト　6-9

```
100 WIDTH 80,25
110 POKE &HE64F,40
120 PRINT "0....5....0....5....0....5....0....5....0"
130 '
140 FOR I=0 TO 50
150   PRINT SQR(I);
160 NEXT
```

実行結果

```
0....5....0....5....0....5....0....5....0
 0  1  1.41421  1.73205  2  2.23607
 2.44949  2.64575  2.82843  3  3.16228
 3.31662  3.4641  3.60555  3.74166
 3.87298  4  4.12311  4.24264  4.3589
 4.47214  4.58258  4.69042  4.79583
 4.89898  5  5.09902  5.19615  5.2915
 5.38517  5.47723  5.56777  5.65686
 5.74456  5.83095  5.91608  6  6.08276
 6.16441  6.245  6.32456  6.40312
 6.48074  6.55744  6.63325  6.70821
 6.78233  6.85566  6.9282  7  7.07107
```

6-6-2　PRINT文とTAB関数

TAB関数の働きが、N-BASICと N88-BASICでは異なります。N-BASICのプログラムをN88-BASICのプログラムに直すとき、注意したいことの1つです。

①値が表示桁数を超える場合

（N88-BASIC）

値を表示桁数で割った余りが、値となります。（ただし、WIDTH文を少なくとも1回実行しておく必要があります。

リスト　6-10(40桁モード)

```
print "A"tab(43)"B" ⏎
A  B
Ok
```

（N-BASIC）

表示桁数に関係なく値の数だけ空白を出力します。

リスト　6-11(40桁モード)

```
print "A"tab(43)"B"⏎
A
  B
Ok
```

②プリント位置がTABの値を超えている場合。

(N88-BASIC)

改行して次の行のTAB位置まで空白を出力します。

リスト　6-12(40桁モード)

```
print "0123456789"tab(7)"**"tab(2)"=="
0123456789
       **
  ==
Ok
```

(N-BASIC)

TABは無効となります。

リスト　6-13(40桁モード)

```
print "0123456789"tab(7)"**"tab(2)"=="⏎
0123456789**==
Ok
```

この他にも、N88-BASICでは、TABの値は-32768～32767までとれますが、N-BASICでは0～255しかとれない、という違いもあります。

6-6-3　PRINT文で矢印を書く

キャラクタコード表を見ると、'↑'とか'C_L'などの直接PRINT文で書けない文字がありますね。特に矢印は使ってみたいものの1つです。PRINT文がだめなら、直接VRAMにPOKEしてしまうのも1つの手ではありますが、VRAMの位置を計算したり、色を付けたりするのがどうも面倒です。

そこで、それらの文字を出力させるテクニックを紹介しましょう。

　　POKE &HE6B6, 1

を実行してみて下さい。

OKのあとに$C_R$$L_F$というように出ましたね。カーソルを移動させると矢印がでます。

もとに戻したければ、次のようにして下さい。

　　POKE &HE6B6, 0

これはプログラム中でも使うことができます。次のプログラムで確かめてみましょう。

リスト　6-14

```
100 WIDTH 40,25
110 POKE &HE6B6,1
120  LOCATE 17,10 : PRINT CHR$(30);
130  LOCATE 16,11 : PRINT CHR$(29);"ロ"CHR$(28);
140  LOCATE 17,12 : PRINT CHR$(31);
150 POKE &HE6B6,0
160 END
```

PRINT文の後に";"がついていますが、これは、'$^{C}{}_{R}{}^{L}{}_{F}$'という文字が出てしまうのを避けるためです。

なお、一文字だけであればCHR$(&H19)のあとに続けて書くと、そのコントロール文字がでます。たとえば、

PRINT　CHR$(&H19);CHR$(&H1C)

とすると右向き矢印が表示されます。

6-7　テキスト画面のGET・PUT

N-BASICでは、テキスト画面のGET(画面のデータを配列に取り込む)、PUT(配列のデータを画面に表示する)が使えましたが、N88-BASICでは、グラフィック画面のGET, PUTしかできません。

ここでは、BASICと機械語を組み合わせてアトリビュートも含めたテキスト画面のGET, PUTサブルーチンを作ってみました。

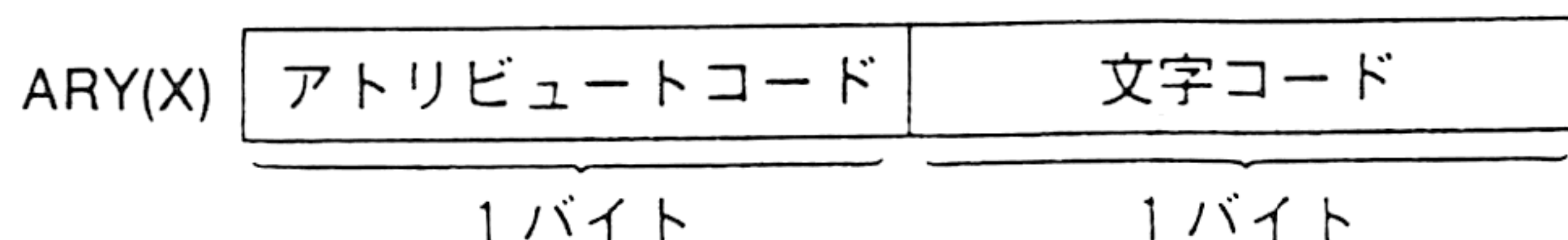
これは、N-BASICでのGET@A, PUT@と同じようなもので、(GX1, GY1)-(GX2, GY2)の間の画面データを整数型配列ARYに読み込んだり、逆に、画面に表示するためのサブルーチンです。

リスト　6-15　テキスト画面GET, PUTサブルーチン

```
1000 '----- GET@ SUB ROUTINE -----
1010 'GET@A(GX1,GY1)-(GX2,GY2),ARY
1020 '----------------------------
1030 '
1040 *GET.SUB
1050  ERASE ARY
1060  GXS=GX2-GX1+1  :  GYS=GY2-GY1+1
1070  DIM ARY(GXS*GYS)
1080  FOR Y=1 TO GYS
1090   FOR X=1 TO GXS
1100     VAD=&HF3C8+120*(GY1+Y-1)+(GX1+X-1)
1110     ARY(GXS*(Y-1)+X)=USR0(VAD)
1120   NEXT X
1130  NEXT Y
1140 RETURN
2000 '----- PUT@ SUB ROUTINE -----
2010 'PUT@A(GX1,GY1)-(GX2,GY2),ARY
2020 '----------------------------
2030 '
2040 *PUT.SUB
2050  GXS=GX2-GX1+1  :  GYS=GY2-GY1+1
2060  FOR Y=1 TO GYS
2070   FOR X=1 TO GXS
2080     POKE &HF2FE,GY1+Y
2090     POKE &HF2FF,GX1+X
2100     DM=USR1(ARY(GXS*(Y-1)+X))
2110   NEXT X
2120  NEXT Y
2130 RETURN
```

＊使用する変数はすべて整数型にしておいて下さい。

配列ARYの１つの要素には、１つの文字コードとアトリビュートコードが入ります。

ARY(X)	アトリビュートコード	文字コード
	1バイト	1バイト

次に簡単なテストプログラムをあげておきます。

リスト　6-16

```
100 '
110 '    GET@A,PUT@A   Sample
120 '
130 '----- INIT -----
140 WIDTH 80,25 : CONSOLE ,,,1
150 CLS
160 DEFINT A-Z
170 DEF USR0=&HF2F0
180 DEF USR1=&HF2E0
190 DIM ARY(0)
200 FOR I=&HF2E0 TO &HF2FC
210   READ D$ : POKE I,VAL("&H"+D$)
220 NEXT
230 DATA 46,23,4E,C5,2A,FE,F2,CD,9D,42,C1,CD,50,43,C9,00
240 DATA E5,7E,23,66,6F,CD,52,44,E1,77,23,71,C9
250 '----- TEST PROGRAM -----
260  GX1=0 : GY1=0
270  GX2=4 : GY2=2
280  COLOR 4 : LOCATE GX1  ,GY1   : PRINT "┌───┐"
290  COLOR 2 : LOCATE GX1+2,GY1   : PRINT "♥";
300  COLOR 4 : LOCATE GX1  ,GY1+1 : PRINT "|   |"
310  COLOR 5 : LOCATE GX1+1,GY1+1 : PRINT "*+*";
320  COLOR 4 : LOCATE GX1  ,GY1+2 : PRINT "└───┘"
330  GOSUB *GET.SUB
340  FOR I=1 TO 20
350    GX1=RND*70 : GY1=RND*20
360    GX2=GX1+4 : GY2=GY1+2
370    GOSUB *PUT.SUB
380  NEXT
390 END
400 '----- GET@ SUB ROUTINE -----
410 'GET@A(GX1,GY1)-(GX2,GY2),ARY
420 '----------------------------
430 '
440 *GET.SUB
450  ERASE ARY
460  GXS=GX2-GX1+1 : GYS=GY2-GY1+1
470  DIM ARY(GXS*GYS)
480  FOR Y=1 TO GYS
490   FOR X=1 TO GXS
500     VAD=&HF3C8+120*(GY1+Y-1)+(GX1+X-1)
510     ARY(GXS*(Y-1)+X)=USR0(VAD)
520   NEXT X
530  NEXT Y
540 RETURN
550 '----- PUT@ SUB ROUTINE -----
560 'PUT@A(GX1,GY1)-(GX2,GY2),ARY
570 '----------------------------
580 '
```

```
590 *PUT.SUB
600  GXS=GX2-GX1+1 : GYS=GY2-GY1+1
610  FOR Y=1 TO GYS
620   FOR X=1 TO GXS
630     POKE &HF2FE,GY1+Y
640     POKE &HF2FF,GX1+X
650     DM=USR1(ARY(GXS*(Y-1)+X))
660   NEXT X
670  NEXT Y
680 RETURN
```

第7章　グラフィック画面

グラフィック画面は次のような流れで表示されます。

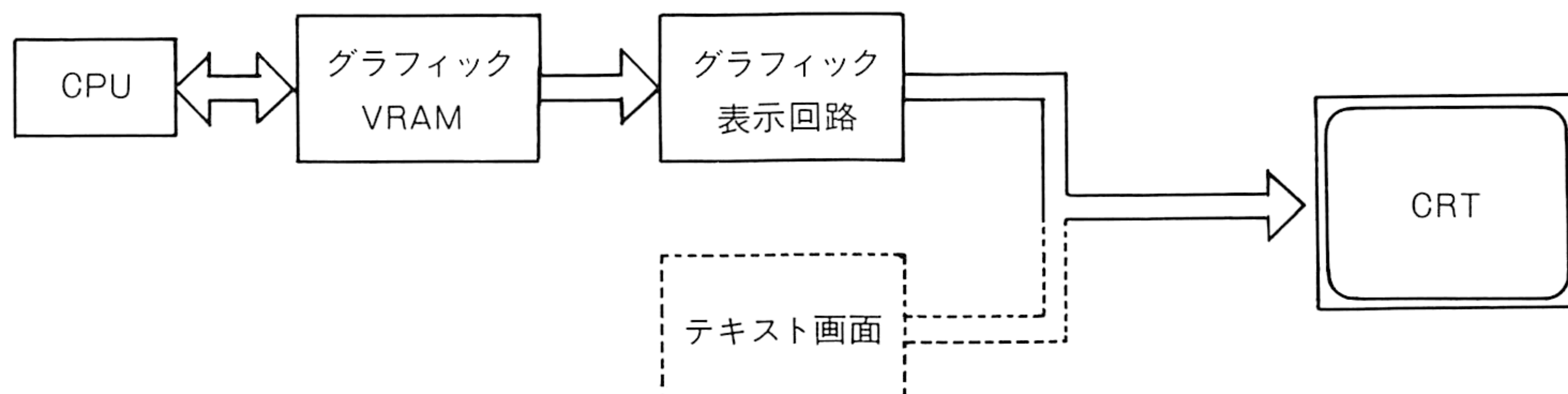



図7-A　グラフィック画面表示の流れ

7-1　グラフィックVRAM

7-1-1　GVRAMとグラフィック画面

PC-8801mkⅡは、グラフィック用のVRAM(GVRAM)としてC000H～FFFFHまでの16Kバイトを3バンク(計48Kバイト)持っています。このGVRAMは、通常CPUのアドレス空間上にはなく、OUT命令でバンクを切り換えてアクセスします。メモリマップは次のようになっています。

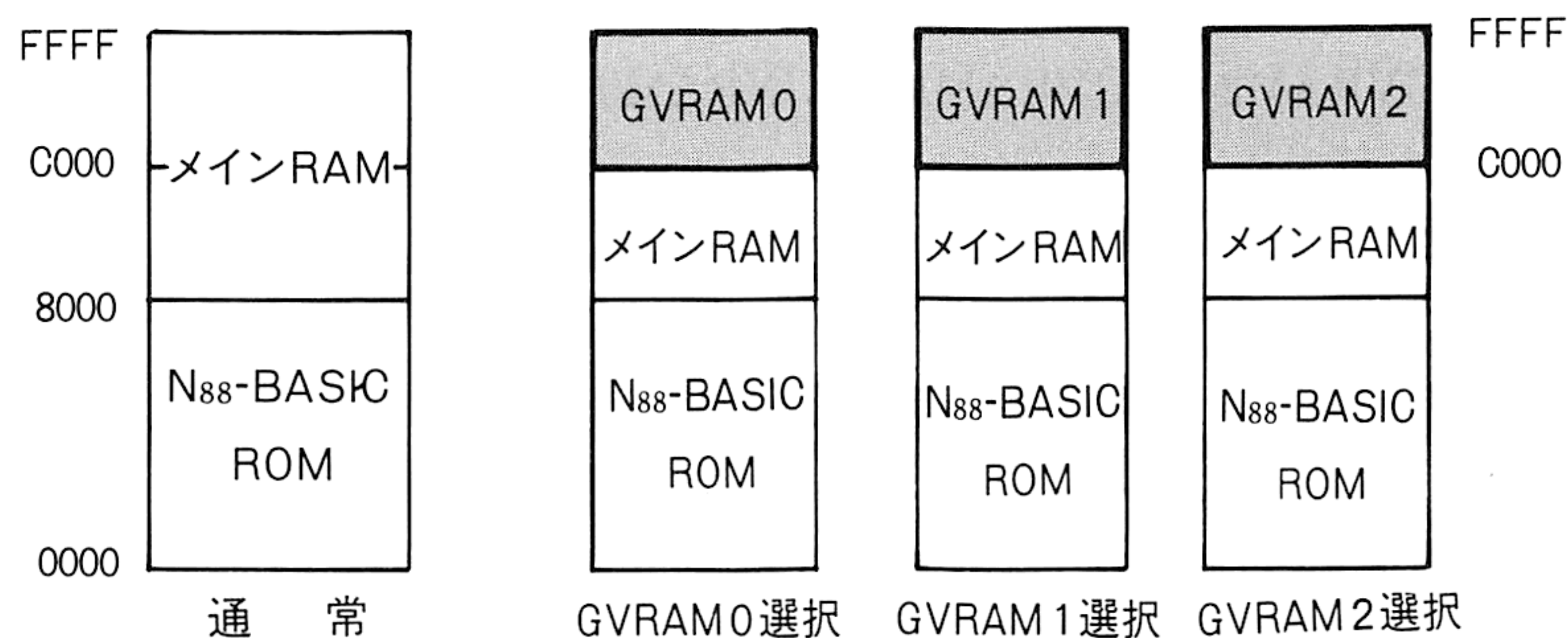



図7-1-A　GVRAMのメモリマップ

各グラフィックモードによって、3つのGVRAMを組み合わせて使っています。

(1)　カラーグラフィックモード(SCREEN　0)

CRTディスプレイの種類にかかわらず、カラーグラフィックの解像度は640×200ドットです。3つのGVRAMを組み合わせて、8種類の色を出しています。カラーパレットを考えない場合は、各GVRAMを光の3原色に割り当てていると思えばよいでしょう。(GVRAM

0＝青，GVRAM 1＝赤，GVRAM 2＝緑）

テキスト画面の色は、このカラーグラフィックとは無関係にCRTC（μPD3301）のアトリビュートで指定できます。また、テキスト画面を表示するか、カラーグラフィック画面を表示するか、あるいは、テキストとカラーグラフィックを同時に表示するかの選択もすることができます。

GVRAMの0～2を重ねあわせたものがCRTの画面に表示されます。各GVRAMの同アドレス、同ビットのデータによってパレットが定まります。パレットの機能については、「6-2」カラーパレットの制御を御覧下さい。

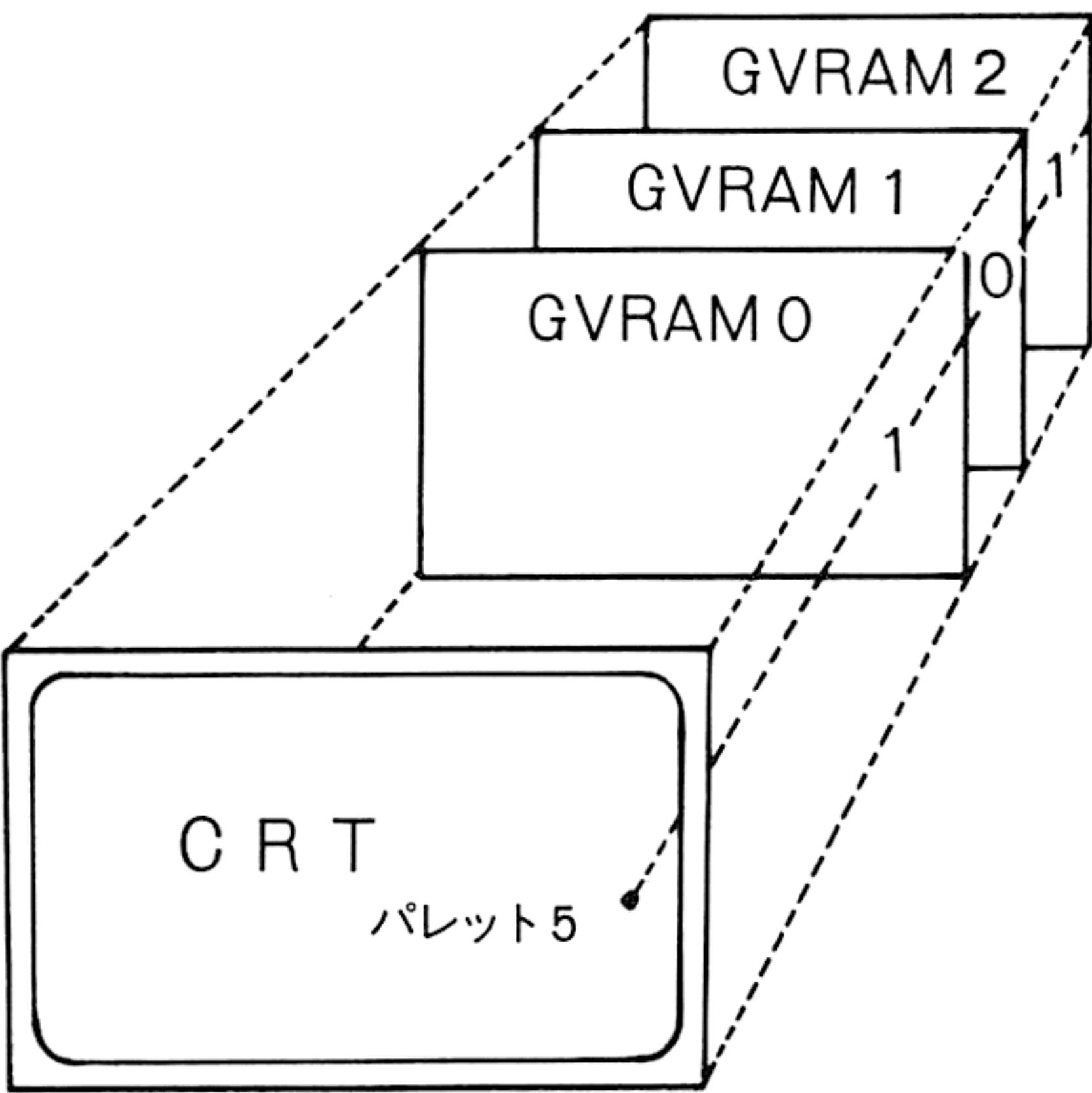



図7-1-B　カラーグラフィックモード

＜各GVRAMのビット状態とパレット番号との対応＞

GVRAM2	GVRAM1	GVRAM0	パレット番号
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

（2） モノクログラフィックモード （SCREEN 1）

グラフィックの解像度は640×200ドットです。このモードでは3つのグラフィックプレーン（GVRAM　0，1，2）があり、各々第1画面～第3画面に割り当てています。これらとテキスト画面と合わせて、4種類の画面ソース（テキスト、GVRAM0，GVRAM 1，

GVRAM 2)を持つことになり、どの画面ソースをも重ね合わせて表示させることができます。この場合、色(アトリビュート)の指定はテキストVRAMのもの、すなわち、μPD3301の機能で行ないますので、最小のアトリビュート指定単位は1キャラクタ(8 × 8 ドット)です。

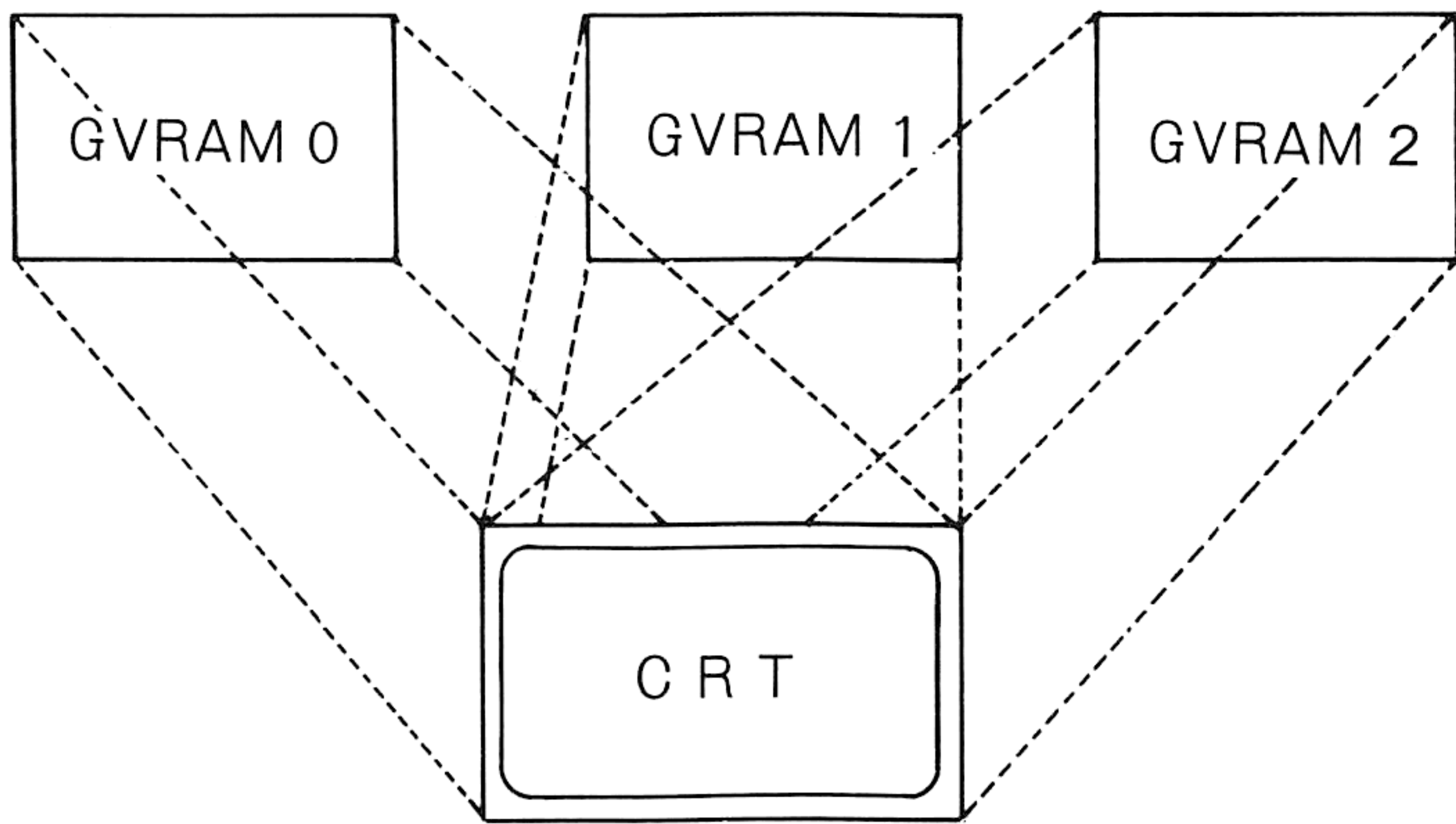



図7-1-C　グラフィックモード

(3)　高解像度グラフィックモード　(SCREEN　2)

これは高解像度CRTが接続されている時にのみ使用できるモードで、解像度は640×400ドットです。このモードでは、グラフィックの表示はGVRAM0, GVRAM1をつなげた 1 ページのみで、画面表示のソースは、このグラフィックとテキストの 2 つです。GVRAM 2 は使用されていません。カラー指定とアトリビュート指定は、グラフィックモードと同じくμPD3301のアトリビュートで指定でき、この場合のアトリビュートを指定できる最小単位は 8 ×16ドットです。

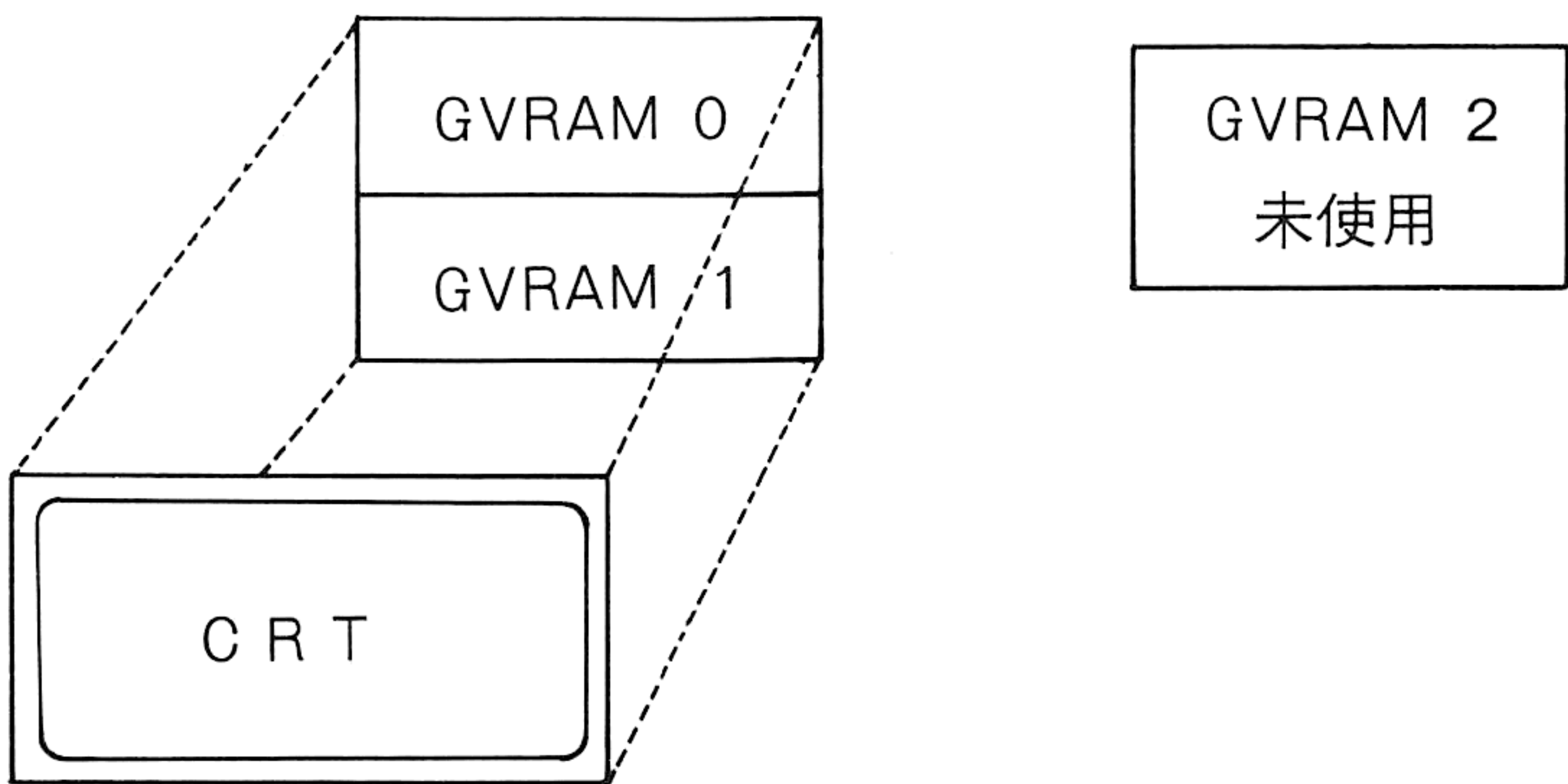



図7-1-D　高解像度グラフィックモード

• GVRAMのアドレス

GVRAMアドレスとグラフィック画面との対応は次のようになっています。

① 640×200モード時のGVRAM(SCREEN 0 or 1)

0 → 639

0 ↓ 199

C000 C001 C04F

C050

GVRAM
0～2

FE30 FE31 FE7F

FE30

MSB LSB

(0, 199) (7, 199)

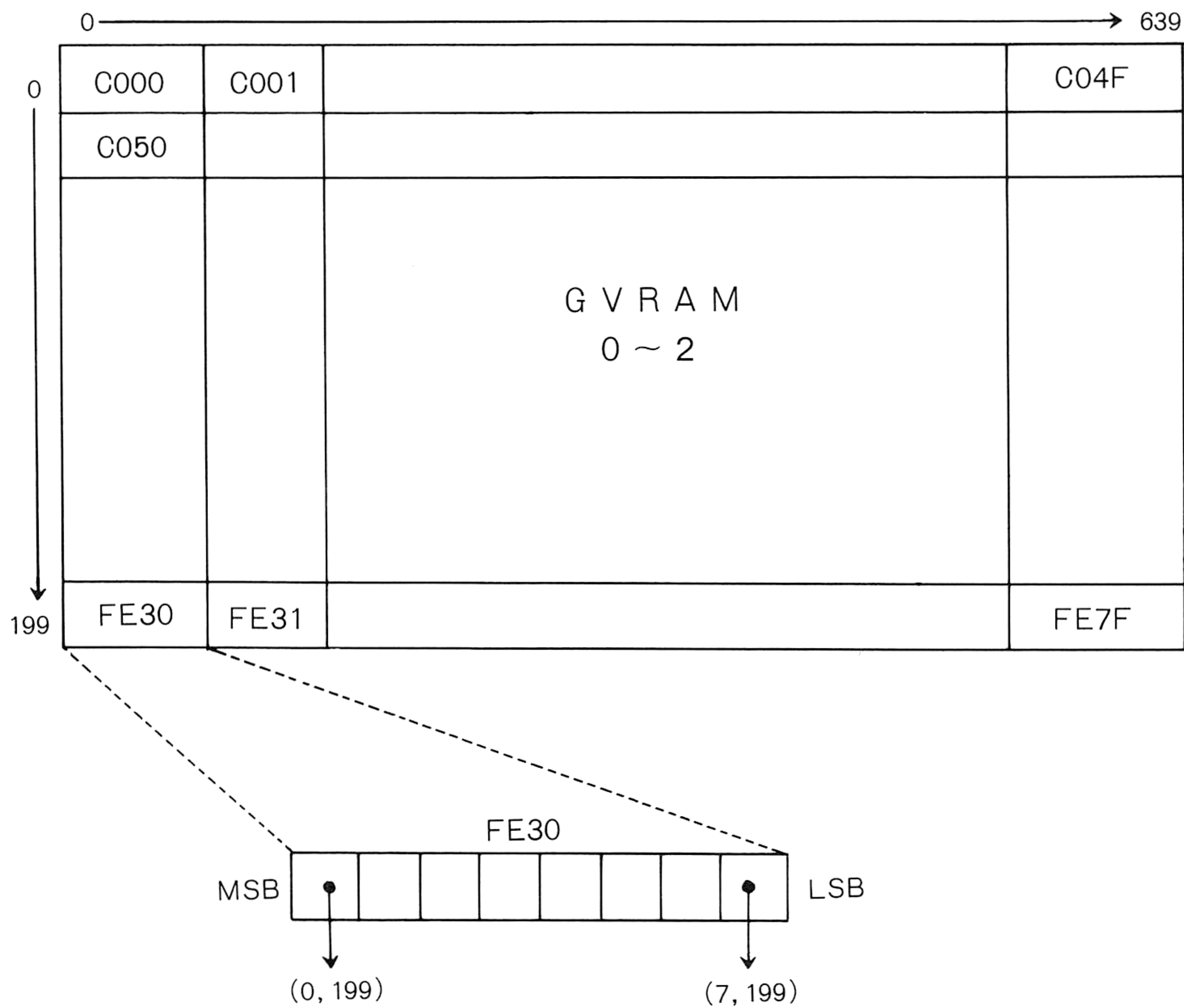



図7-1-E 640×200モード時のGVRAM

② 640×400モード時のGVRAM(SCREEN 2)

0 — 639

0 — 199, 200 — 399

C000 C04F

C050

GVRAM 0

FE30 FE7F

C000 C04F

GVRAM 1

FDE0 FE2F

FE30 FE7F

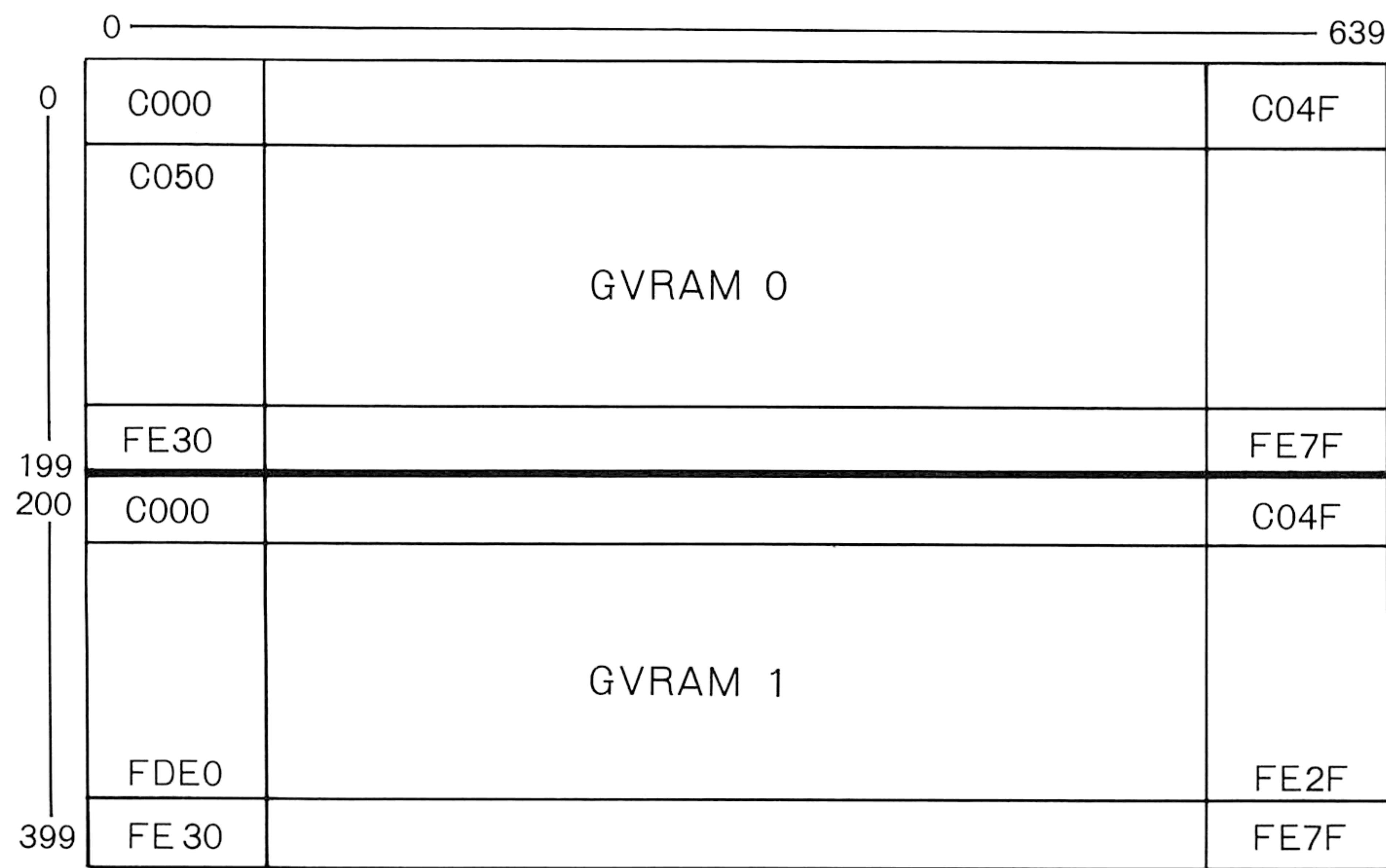



図7-1-F 640×400モード時のGVRAM

たとえば、ウインドウもビューポートも考えない場合は、点(100, 150)はアドレス150×80＋100￥８＋C000H ＝ EEECH、ビットは100 mod 8 ＝ 4ですから、左から５個目です。第２章の拡張PEEK／POKEプログラム(P30)を使ってやってみましょう。

SCREEN 0 ： ROLL 197

FOR I=0 TO 7 ： COLOR=(I, I) ： NEXT

として画面をイニシャライズし

POKE &HEEEC, &H8," 2"

とすると、画面に緑色の点が現われますね。これが本当に(100, 150)にあるかどうか確かめるために、

PSET(100, 150), 2

とすると、緑色の点が赤色になりました。間違いなかったことがわかりました。

ここでGVRAMを直接アクセスするサンプルプログラムを上げておきます。拡張PEEK／POKEを使っています。

リスト 7-1

```
100 DEFINT A-Z
110 SCREEN 0 : CONSOLE ,,0 : CLS
120 FOR I=0 TO 7 : COLOR=(I,I) : NEXT
130 '
140 FOR I=0 TO &H3FFF
150   D=PEEK(I+&H1000)
160   POKE &HC000+I,D,"2"
170 NEXT
180 '
190 FOR I=0 TO 500
200   PAINT(RND*639,RND*199),3,4
210 NEXT
```

7-1-2 GVRAMのアクセス

すでに述べたようにGVRAMはメインRAMと同じアドレスにあって、バンク切り換えを行なってから読み書きします。

GVRAMをセレクトするためには次のようにします。

GVRAM0……OUT 5CHを行なう。

GVRAM1……OUT 5DHを行なう。

GVRAM2……OUT 5EHを行なう。

＊OUTするデータは何でもよい。

こうするとアドレスC000H～FFFFHに選んだGVRAMが現れ、C000H～FFFFHのメインRAMはアクセスできなくなります。再び、メインRAMをアクセスするためには、

OUT 5FH

を行ないます。これもOUTするデータは何でもかまいません。

残念ながらこれらの制御はBASICで行なうことはできません。BASICはC000H～FFFFHのメインRAMを必要とするからです。そこで機械語を使って行なうわけですが、この時にいくつか注意しなければならないことがあります。

(1)GVRAMをアクセスする機械語自身はC000H番地よりも前に置く。

そうしないとGVRAMをセレクトした瞬間にその機械語自身がCPUから見えなくなってしまいます。

(2)GVRAMをセレクトしている間はインタラプトを止めておく。

N88-BASICではインタラプトの処理のためにC000H〜FFFFHのメインRAMを使うからです。独自にインタラプトテーブルやインテラプト処理ルーチンを作成した場合にはこの限りではありませんが。

(3)GVRAMをセレクトしている間は、スタックを使わない。(PUSH，POP，CALL，RET，RSTなどを使わない。)

N88-BASICから機械語を使用するときは、通常スタックはメインRAMのC000Hよりも後にあります。したがってGVRAMをセレクトすると、スタックは使用できなくなります。スタックを使用したい時には、スタックをC000Hより前に設定する必要があります。これは①GVRAMをセレクトする前にスタックを移し、再びメインRAMをセレクトしてからスタックをもとに戻すか、②BASICレベルでCLEAR，&HBFFFを行なうかします。どちらの方法をとるにしても、メインRAMでC000Hより前で使用可能な分は非常にせまいので、必要な部分(変数エリアやファイルテーブル、ディスクコードなど)をこわさないようにして下さい。どの部分が使用可能かは第3章N88-BASICの内部構造を参照して確認してください。

(4)GVRAMをセレクトしている時間は短くする。

フラッシュモード(後述)でない場合はGVRAMをセレクトするとCPUの速度が低下します。グラフィック画面を表示中は実行が止まるようになり、帰線中にのみ実行が行なわれるからです。

GVRAMをアクセスする一般的なプログラムは次のような形になります。

```
            DI
            OUT  (5?H), A
  GVRAMの読み書き
            OUT  (5FH), A
            EI
```

たとえば、GVARAMに1バイト書き込むプログラムを考えて見ましょう。

HLに書き込みたいアドレス(C000H〜FFFFH)
Aに書き込みたいデータ
Cに書き込みたいGVRAM(5C，5D，5E)

とすると

```
            DI
            OUT  (C), A
            LD   (HL), A
            OUT  (5FH), A
            EI
```

となります。

また、現在どのGVRAMがセレクトされているかを知ることができます。これにはポート5CHを使います。

ポート 5CH (IN)

7	6	5	4	3	2	1	0
/	/	/	/	/	G2	G1	G0

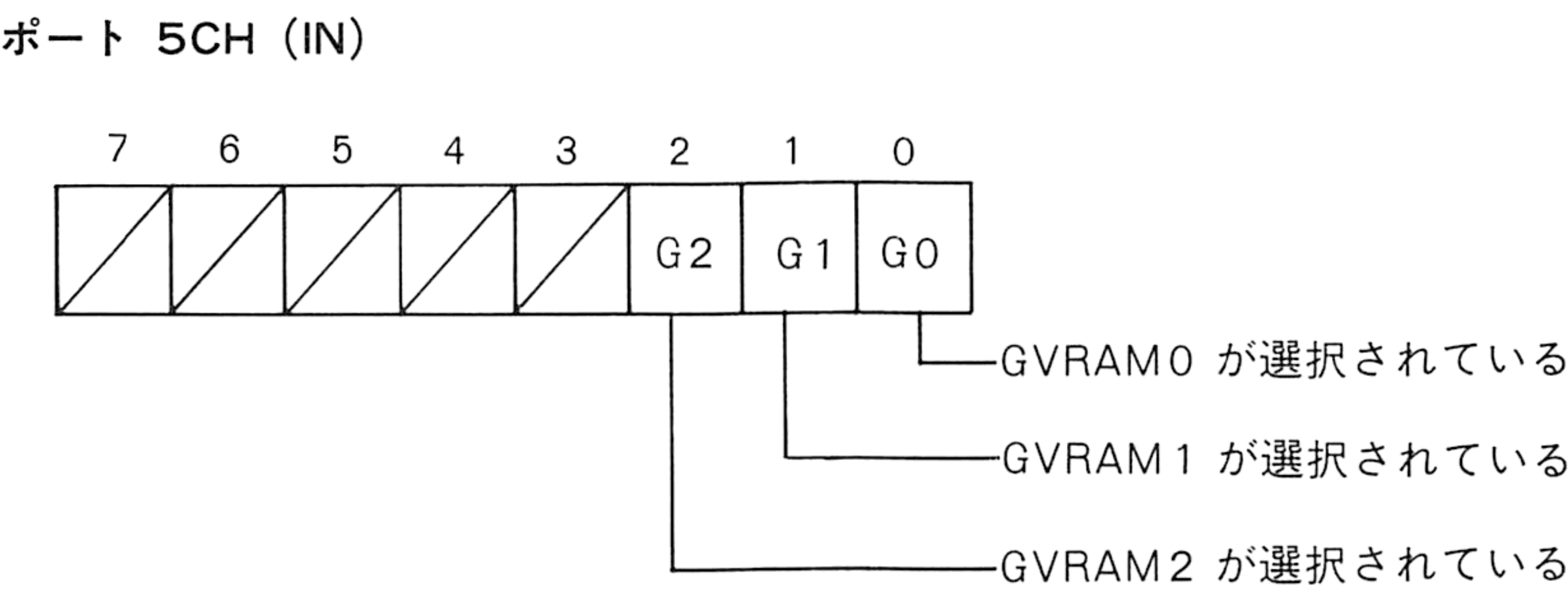



図7-1-G GVRAMステータスポート

7-1-3 カラーグラフィック画面のセーブ、ロード

グラフィック画面をディスクにセーブすることを考えましょう。GVRAMはメインRAM上にありませんから、残念ながらBSAVEでセーブすることはできません。そこでデータファイルとして書き出すことにします。ファイル形式はランダムアクセスファイルとして、次のようなフォーマットで格納します。

Blue	Red	Green
レコード#1	レコード#2	レコード#3
レコード#4	レコード#5	レコード#6
	≀	
レコード#187	レコード#188	レコード#189

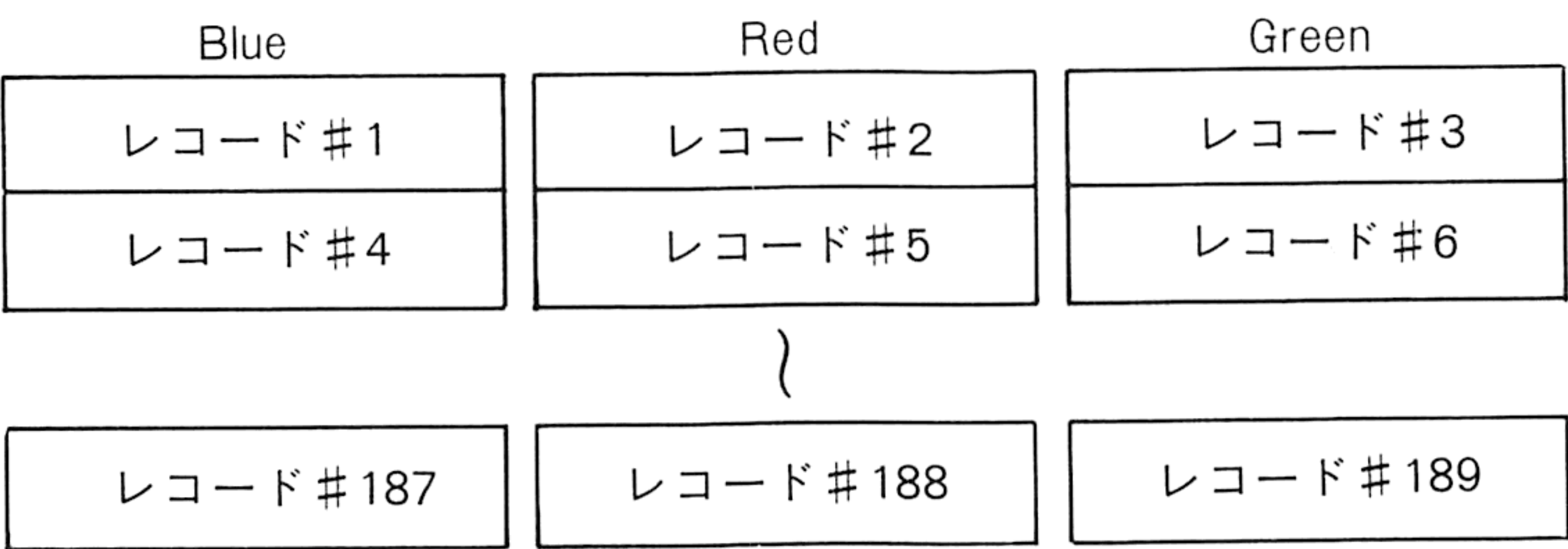



図7-1-H グラフィック画面格納フォーマット

下のがそのプログラムです。機械語の部分は引き数で指定したアドレスから、一度に256バイトをGVRAMから読んで、I／Oバッファに直接書き込みます。そうしてPUT文でディスクに書き出しているわけです。セーブするファイル名は140行のOPEN文で指定します。必要があれば変えてください。

リスト 7-2

```
100 '
110 '     GRAPHIC SCREEN SAVE
120 '
130 DEFINT A-Z
140 OPEN "GVRAM.scr" AS#1
150 '--------- Write machine code
160 BUF0=VARPTR(#0)+9 : BUF1=VARPTR(#1)+9
170 RESTORE *SDATA
180 FOR I=0 TO 31
```

```
190   READ D$ : POKE BUF0+I,VAL("&H"+D$)
200 NEXT
210 *SDATA
220 DATA 0E,5C,18,06,0E,5D,18,02,0E,5E,2A,41,EC,11,55,55
230 DATA F3,ED,79,01,00,01,ED,B0,D3,5F,FB,C9,00,00,00,00
240 POKE BUF0+&HE,PEEK(VARPTR(BUF1))
250 POKE BUF0+&HF,PEEK(VARPTR(BUF1)+1)
260 DEF USR0 = BUF0 : DEF USR1 = BUF0+4 : DEF USR2 = BUF0+8
270 '-------- GVRAM address set
280 PAGE = 189 : AD = &HFE00
290 '-------- Move GVRAM into DISK
300 FOR I = 63 TO 1 STEP -1
310   DMY=USR2(AD) : PUT#1,PAGE : PAGE=PAGE-1   :' Green
320   DMY=USR1(AD) : PUT#1,PAGE : PAGE=PAGE-1   :' Red
330   DMY=USR0(AD) : PUT#1,PAGE : PAGE=PAGE-1   :' Blue
340   AD = AD - &H100
350 NEXT
360 CLOSE
370 END
```

次はGVRAMにロードします。これはセーブのときの逆をやればよいわけで、考え方は同じです。

リスト 7-3

```
100 '
110 '    GRAPHIC SCREEN LOAD
120 '
130 DEFINT A-Z
140 OPEN "GVRAM.scr" AS#1
150 SCREEN 0,0
160 '-------- Write machine code
170 BUF0=VARPTR(#0)+9 : BUF1=VARPTR(#1)+9
180 RESTORE *SDATA
190 FOR I=0 TO 31
200   READ D$ : POKE BUF0+I,VAL("&H"+D$)
210 NEXT
220 *SDATA
230 DATA 0E,5C,18,06,0E,5D,18,02,0E,5E,2A,41,EC,11,55,55
240 DATA EB,F3,ED,79,01,00,01,ED,B0,D3,5F,FB,C9,00,00,00
250 POKE BUF0+&HE,PEEK(VARPTR(BUF1))
260 POKE BUF0+&HF,PEEK(VARPTR(BUF1)+1)
270 DEF USR0 = BUF0 : DEF USR1 = BUF0+4 : DEF USR2 = BUF0+8
280 '-------- GVRAM address set
290 AD = &HC000
300 '-------- Move GVRAM into DISK
310 FOR I = 1 TO 63
320   GET#1 : DMY=USR0(AD) :' Blue
330   GET#1 : DMY=USR1(AD) :' Red
340   GET#1 : DMY=USR2(AD) :' Green
350   AD = AD + &H100
360 NEXT
370 CLOSE
380 END
```

このプログラムは大変遅くて、ロードするのに数十秒もかかります。これを改善するには、ディスクからの読み込みをセクタ単位でなくクラスタ単位で行なう、データを圧縮してセーブしておく、などの方法がありますので考えてみてください。

7-2　カラーパレット

7-2-1　カラーパレットの制御

カラーグラフィックモードでは、GVRAM０，１，２の同アドレス、同ビットに書かれているデータとカラーパレットにより表示の色を指定します。GVRAMのデータとパレット番号の対応はこうなっています。

表７-２-１　GVRAMのデータとパレット番号の対応

パレット番号	GVRAM2	GVRAM1	GVRAM0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

各パレットに、８色のうち１色を割りあてて色を出しますが、色の割り当ては自由にできます。N88-BASICではイニシャライズ時に表 ７-２-２ のように割り当てます。

表７-２-２　イニシャライズ時のパレット

パレット番号	0	1	2	3	4	5	6	7
色	黒	青	赤	紫	緑	水色	黄	白

GVRAMのアドレスC000Hのビット7に、GVRAM0は１、GVRAM1は０、GVRAM2は１が書かれているとします。パレット番号は表７-２-１より５となりますので、パレットをN88-BASICのイニシャライズ時のものと変えていなければ、CRTの左上隅に水色のドットが出ます。このパレット番号と色の割り当てはOUT命令で変えることができます。

表７-２-３　カラーパレットの設定

ポートアドレス	パレット番号
OUT　54H	0
OUT　55H	1
OUT　56H	2
OUT　57H	3
OUT　58H	4
OUT　59H	5
OUT　54H	6
OUT　58H	7

上の各ポートにカラーコードを出力すると、対応するパレットに色が設定されます。データの上位５ビットは無視されます。

カラーコード	0	1	2	3	4	5	6	7
色	黒	青	赤	紫	緑	水色	黄	白

サンプルプログラムで説明しましょう。

リスト　7-4

```
10 SCREEN 0
20 FOR I=0 TO 7 : COLOR=(I,I) : NEXT
30 '
40 FOR I=0 TO 7
50   LINE(0,I*20)-STEP(639,19),I,BF
60 NEXT
```

このプログラムで8色の帯をCRTに表示させます。OUT命令でパレットをコントロールすると、色を瞬時に変えることができます。

OUT　&H56, 6⏎　　　　'COLOR=(2, 6)と同じ

とすると赤色の帯が黄色になります。

7-3　グラフィックモードの設定

グラフィックモードの切り換えはI／Oポート31Hを使って行なわれます。

ポート 31H (OUT)　　N88-BASICのワークエリア：E6C2H番地

7	6	5	4	3	2	1	0
／	／	25LINE	HCOLOR	GRPH	RMODE	MMODE	200LINE

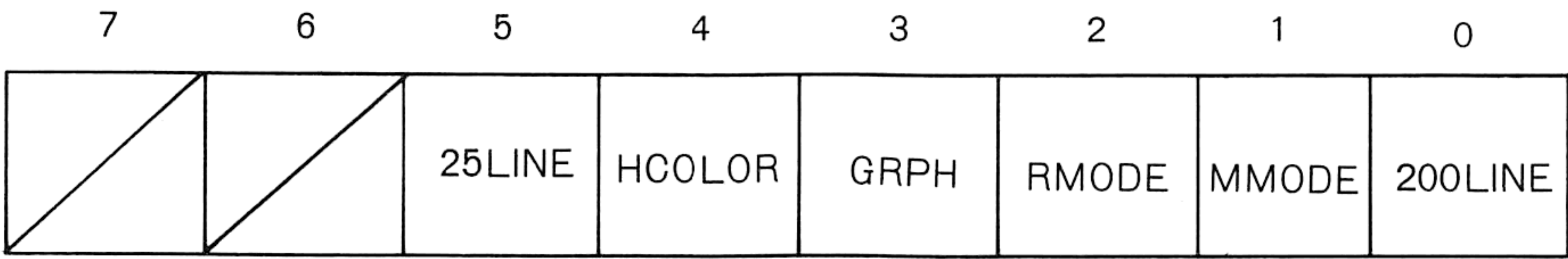



図7-3-A　グラフィックモード切り換え

これらのうち、HCOLORとGRAPHの2つのビットでグラフィックモードを切り換えます。

HCOLOR	GRAPH	グラフィックモード
0	0	テキストモード
0	1	モノクログラフィックモード
1	0	禁止
1	1	カラーグラフィックモード

高解像度CRTの場合はさらに2つのビットが意味を持っています。

200line	モノクログラフィックモードの切り換え
0	640×400ドット　モード
1	640×200ドット　モード

25line	
1	高解像度CRTの25行モード
0	上記以外の全てのモード

このポートはメモリモードの切り換えにも使われていますから、そのビットを変えないようにしなければなりません。したがって、メモリモードの切り換えと同じく、ワークエリアを参照して出力する値を決定します。N88-BASICでは次のようにします。

```
10  A%=PEEK(&HE6C2)
20  A%=A%  AND  6       'メモリモード以外のビットを0にする
30  A%=A%  OR  (グラフィックモードによる値)
40  POKE  &HE6C2, A%
50  OUT  &H31, A%
```

7-4　高速グラフィックアクセス

PC-8801mkⅡでは合計48Kバイトものグラフィックメモリを扱うために、グラフィック関係の命令が遅くなっています。そこでこれを少しでも早くするテクニックを紹介します。

7-4-1　高速書き込みモード

SCREEN文の第2パラメータによって高速書き込みモードにすることができます。GVRAM書き込みは通常は帰線中にのみ行なわれますが、このモードにすると、いつでも書き込めるようになります。画面にチラツキが出ますが、2〜3倍の速度で描画できます。このモードはI／Oポート40Hでコントロールされています。なお、このポートも、他のビットは使われているので注意してください。

ポート 40H（OUT）　N_{88}-BASICのワークエリア：E6C1H番地

7	6	5	4	3	2	1	0
UOP2	UOP1	BEEP	FLASH	$\overline{CLPS}$	CCK	CSTB	$\overline{PSTB}$

FLASH
- 0：ノーマルモード
- 1：フラッシングモード

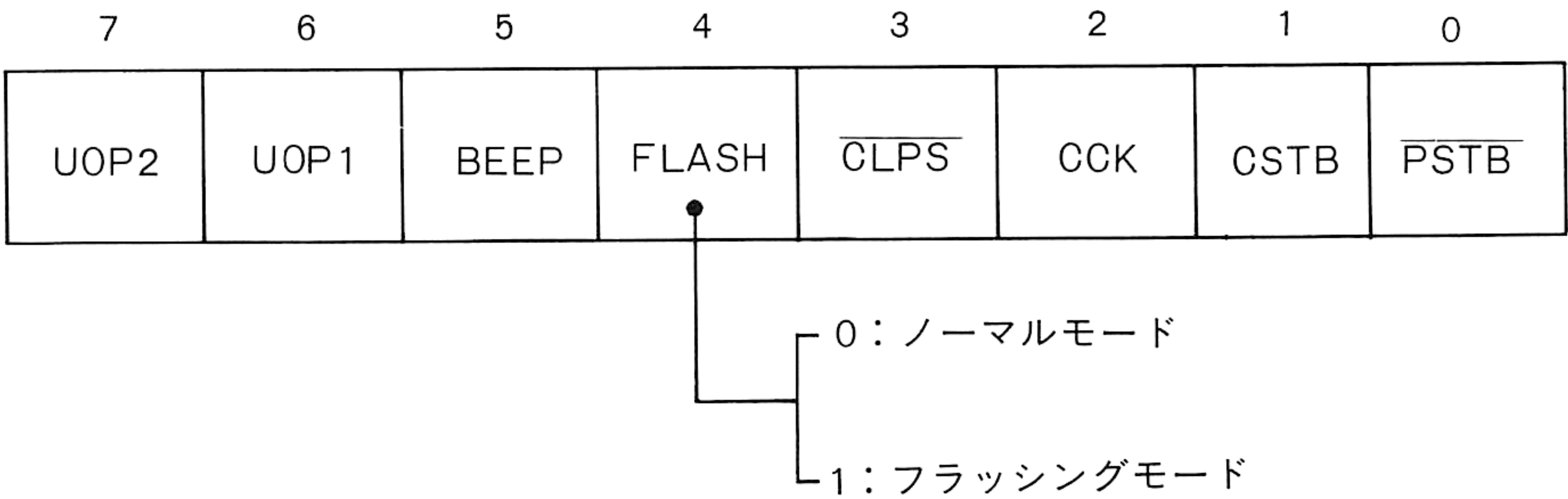



図7-4-A　フラッシングモード

7-4-2　CPUの実行速度

CPUの実行速度を上げると当然グラフィック描画も速くなります。これには次のような方法があります。

①テキスト画面のDMAを止める。

拡張命令のCMD　TEXT　OFFによってDMAを止めることができます。詳しくは「第6章-4　DMAC」を御覧ください。

②キースキャンを止める。

POKE & HE6CD, 255でキースキャンを止めることができます。詳しくは「第5章-2　キースキャン」を御覧ください。

7-4-3　高速画面クリア

拡張命令を使用しているときにはCMD CLS 2を行なえばよいわけですが、そうでない場合や、PC-8801と互換性をとりたいときには次の方法をとります。

①まず、CLS　2よりもROLL文を使う方が速いということがあげられます。これはCLS 2はVIEWウインドウ内だけをクリアするので、VIEWウインドウを計算する分遅くなるからです。

実際、CLS 2はBOX FILLのルーチンを用いて行なわれています。これに対してROLL文はVI EWウインドウを無視してGVRAMの内容を直接に移動するだけです。ROLL文では一度に197ドットしかスクロールできませんから繰り返して行ないます。

・640×200のとき

ROLL 197：ROLL 197

・640×400のとき

ROLL 197：ROLL 197：ROLL 197

②次に、高速アクセスモードにするか、グラフィック表示を消します。つまり、SCREEN，1～SCREEN， 3で行なうわけです。特にSCREEN，3でグラフィック表示を消してから画面消去を行ない、SCREEN 0で表示を戻すとチラツキもなくきれいに消えます。

BASICで行なえるのはここまでです。これ以上早くするには機械語を使います。

③機械語を用いる場合でも、テクニックを使ってなるべく早くします。鍵となるのはPUSH命令です。これは一度に2バイトを書き込めますので、高速でクリアできます。ここではダンプリストだけをのせておきます。ソースリストは付録(P432)を見てください。また、この場合も当然高速アクセスモードにした方が速くなります。

この例ではBF00H番地からになっていますが、このプログラムはリロケータブルですので、C000H番地よりも前でしたらどこにでも置けます。使わないI／Oバッファに置くのもよいかもしれません。

リスト 7-5

```
BF00 21 00 00 39 11 00 00 0E 5C F3 31 80 FE 3E 02 ED
BF10 79 06 FA D5 D5 D5 D5 D5 D5 D5 D5 D5 D5 D5 D5 D5
BF20 D5 D5 D5 10 EE 3D 20 E9 0C 79 FE 5F 38 DC D3 5F
BF30 F9 FB C9 00 00 00 00 00 00 00 00 00 00 00 00 00
```

7-4-4 グラフィック・データ高速書き込みサブルーチン

GVRAMにデータを書くには、N88-BASICのグラフィック命令(PSET，LINEなど)を使いますが、処理が遅いために、ゲームなどに用いるにはいまひとつです。ここでは、ある決まったパターンを高速に書き込む機械語サブルーチンを紹介します。

リスト 7-6 グラフィック・データ書き込みサブルーチン

```
866A 00 7E 23 66 6F ED 5B 60 F2 EB 4E 23 46 23 EB C5
867A F3 1A D3 5C 77 D3 5F 13 1A D3 5D 77 D3 5F 13 1A
868A D3 5E 77 D3 5F 13 FB 23 10 E6 C1 C5 3E 50 90 4F
869A 06 00 09 C1 0D 20 D8 C9
```

このサブルーチンもリロケータブルになっていますので、RAM上のC000H番地より前ならどこにでも置くことができます。上の例は、N88DISK-BASICの場合で、モニタのコマンドメッセージの部分を使いました。N88ROM-BASICでは、ファイルバッファ#0のところにでも置くとよいでしょう。このサブルーチンで書き込むデータの形式は次のようになっています。

・データの形式

```
                                        ┌────────────────1バイトめのGVRAM0のデータ
                                        │   ┌────────────        //      1  //
                                        │   │   ┌────────        //      2  //
××△△  [ Y ]  [ X ]  [G0][G1][G2]----------------[G0][G1][G2]
         │      │     └──────────────┬──────────────────┘
         │      │                 (X×Y組)
         │      └───── 横のバイト数（ドット数ではない）
         └──────────── 縦のドット数

F260 [△△] [××]  データの格納されている先頭番地
```

例）横16ドット，縦4ドットのパターンのデータをE000Hから入れる場合

16ドット / 4ドット

Data1	Data5
Data2	Data6
Data3	Data7
Data4	Data8

```
E000  04  02  ×× ×× ××   ×× ×× ××  ……  ×× ×× ××
              Data1        Data2         Data8
F260  00  E0
```

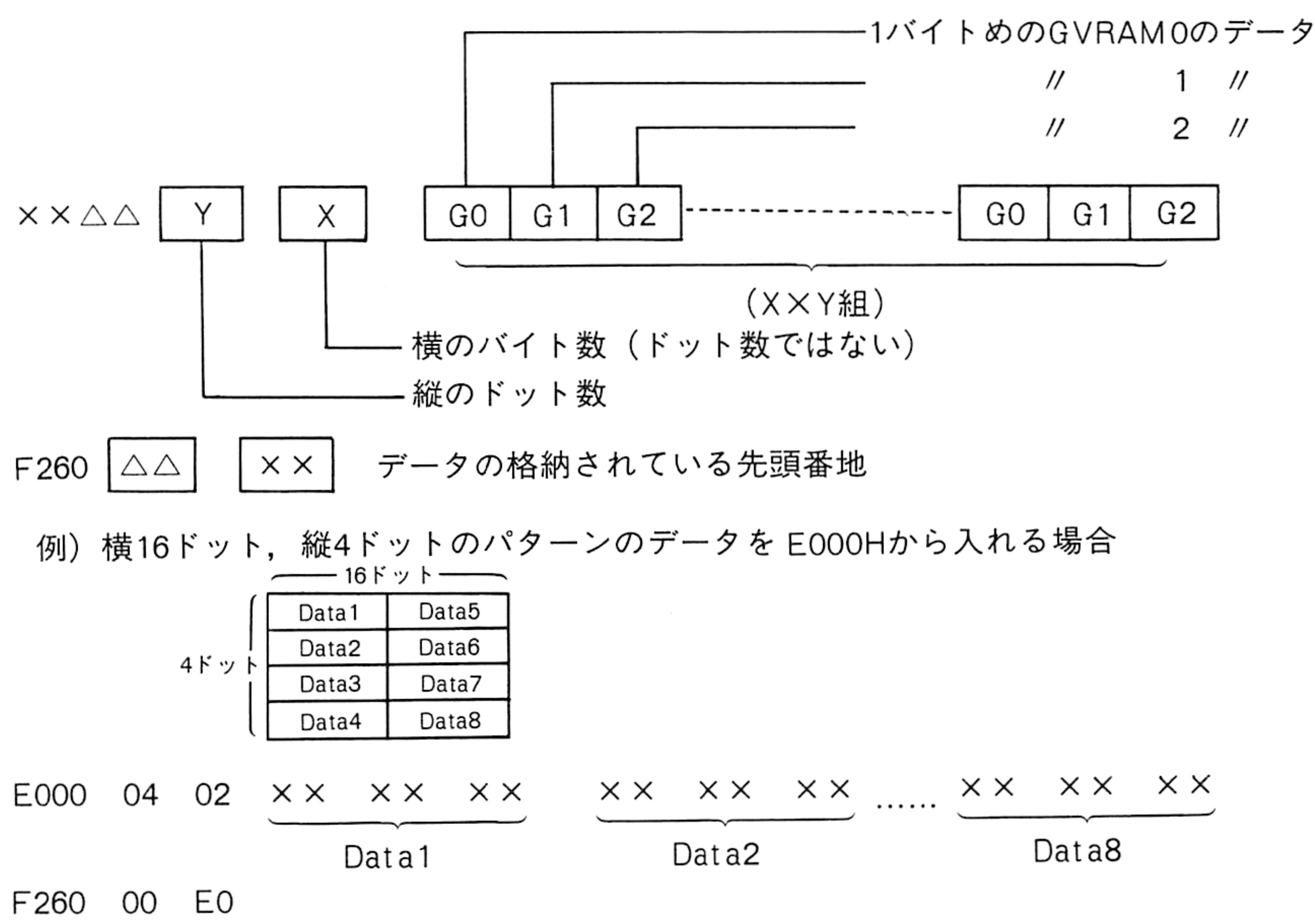



図7-4-B　グラフィックデータ高速書き込みデータ形式

【使い方】

グラフィック画面はカラーモードにしておいて下さい。(SCREEN　0)

DEF　USR＝&H866A　(機械語サブルーチンの先頭)

でUSR関数を定義しておきます。あとは、表示したい位置のGVRAMのアドレスを引数として、USR関数を使えば、データが表示されます。

DUMMY＝USR(&HC000)

引数は整数型でなければなりません。

それでは実際に使ってみましょう。先のサブルーチンが正しく入力されていることを確認して、次のデータを入力します。

(CLEAR，&HDFFFを行なっておくこと)

リスト　7-7

```
E000 10 04 00 FF FF 00 FF FF 00 FF FF 00 F0 F0 00 FF
E010 FF 00 FF FF 00 FF FF 00 F0 F0 00 FF FF 00 FF FF
E020 00 FF FF 00 F0 F0 00 FF FF 00 FF FF 00 FF FF 00
E030 F0 F0 00 86 86 00 0C 0C 00 18 18 00 30 30 01 79
E040 01 F0 F3 F0 07 E7 07 C0 C0 C0 07 F7 07 E0 EF E0
E050 1F DF 1F 80 B0 80 0F EF 0F C0 DF C0 3F BF 3F 00
E060 70 00 1F DF 1F 80 BF 80 7E 7E 7E 00 F0 00 3F BF
E070 3F 00 7E 00 FC FD FC 00 F0 00 7C 7C 7C 01 F9 01
E080 F0 F3 F0 00 E0 00 00 83 83 00 06 06 00 0C 0C 00
E090 10 10 00 FF FF 00 FF FF 00 FF FF 00 F0 F0 00 FF
E0A0 FF 00 FF FF 00 FF FF 00 F0 F0 00 FF FF 00 FF FF
E0B0 00 FF FF 00 F0 F0 00 FF FF 00 FF FF 00 FF FF 00
E0C0 F0 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00


F260 00 E0
```

DEF　USR＝&H866A

DUMMY　＝USR(&HC000)

を実行すると画面左上になにやら変なマークが出ましたね。

機械語で使うときは次のようにします。

HL＝GVRAMの表示するアドレス

DE＝データ格納アドレス

を設定した後

CALL　8673H

このサブルーチンはカラーモード用ですので、白黒モードで使う場合には、不要なバンクのデータを0にしておくとよいでしょう。

機械語に自身のある方は、ソースリスト(付録-1)を参考に白黒モード専用サブルーチンを作ってみて下さい。

＜グラフィック・データ・ジェネレータ＞

このグラフィック・データ書き込みサブルーチンはどうですか。高速なのは良いのですが、データを作るのが面倒ですね。そこで、このデータを作成するプログラムを作ってみました。先程のデータも、このプログラムを使って作成したものです。

このプログラムでは、GVRAMのデータを読むのに、第2章の拡張PEEK文を使っています。810行でエラーが出る場合は、拡張PEEK文(アドレスE000H〜の方)のプログラムを実行して下さい。

まず、作成したいパターンの大きさを入力します。(横48、縦24が最大です)あとは、カーソルをテンキーで操作してパターンを作ります。

終わったら、しばらくたってデータが入っているアドレスが表示されますので、必要ならばその範囲のデータをセーブしておいて下さい。

N88DISK-BASICの場合は、920行の先頭の ’ を除くと自動的にセーブされます。

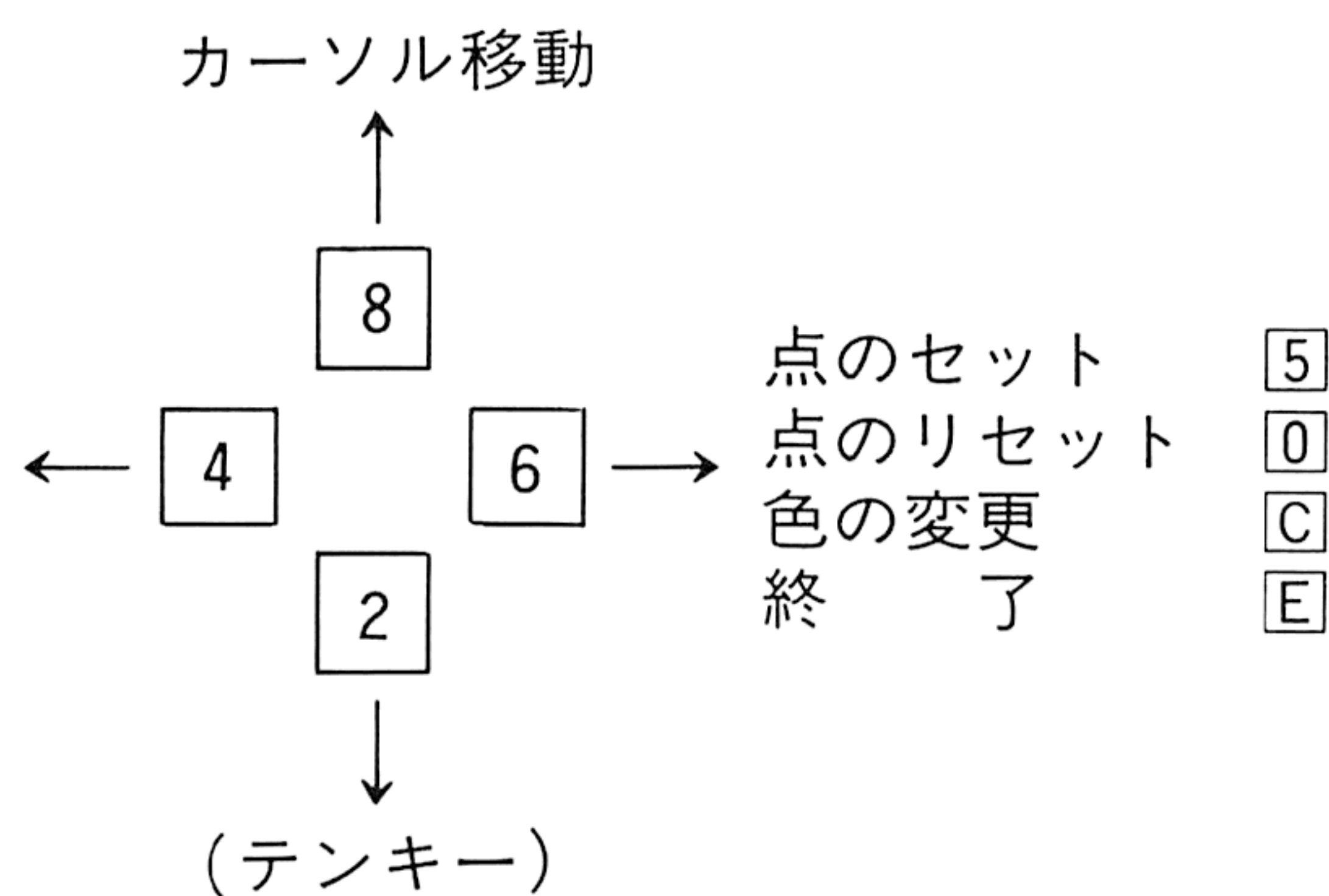



図7-4-C　グラフィック・データ・ジェネレータ　キー操作

リスト 7-8

```
100 '
110 '  Graphic Data Generator
120 '
130 CLEAR ,&HDFFF
140 CONSOLE 0,25,0,1 : WIDTH 80,25
150 COLOR 7
160 SCREEN 0,3 : ROLL 197 : ROLL 3 : SCREEN 0,0
170 DEFINT A-Z
180 '
190  INPUT "INPUT MAX.X(dot),MAX.Y(dot)";MAX.X,MAX.Y
200  IF MAX.X<1 OR MAX.X>48 THEN 190
210  IF MAX.Y<1 OR MAX.Y>24 THEN 190
220 '
230  MAX.BX=(MAX.X-1)¥8 : MAX.BY=MAX.Y-1
240 '
250  CLS
260  LOCATE 23,0
270  FOR X=1 TO MAX.X
280    PRINT HEX$(X MOD 10);
290  NEXT X
300  FOR Y=1 TO MAX.Y
310    LOCATE 20,Y : PRINT USING "##:";Y;
320  NEXT Y
330  LINE(182,7)-(183+MAX.X*8,8+MAX.Y*8),7,B
340  CUR.X=23 : CUR.Y=1
350  COL=7
360 ' --- main loop ---
370 *MAIN
380  LOCATE CUR.X,CUR.Y
390  IN$=INPUT$(1)
400  ON INSTR("246850",IN$) GOSUB *DW,*LF,*RT,*UP,*ST,*RS
410   IF IN$="E" OR IN$="e" THEN *GN.END
420   IF IN$="C" OR IN$="c" THEN GOSUB *CH.COLOR
430  GOTO *MAIN
440 ' --- Cursol move ---
450 *DW
460  IF CUR.Y<MAX.Y     THEN CUR.Y=CUR.Y+1
470 RETURN
480 *LF
490  IF CUR.X>23        THEN CUR.X=CUR.X-1
500 RETURN
510 *RT
520  IF CUR.X<MAX.X+22 THEN CUR.X=CUR.X+1
530 RETURN
540 *UP
550  IF CUR.Y>1         THEN CUR.Y=CUR.Y-1
560 RETURN
570 '
580 *ST
590  PRINT "●"; : PSET(CUR.X-23,CUR.Y-1),COL
600 RETURN
610 *RS
620  PRINT " "; : PSET(CUR.X-23,CUR.Y-1),0
630 RETURN
640 ' --- change color ---
650 *CH.COLOR
660  LOCATE 0,20 : PRINT "COLOR ?";
670  CL$=INPUT$(1)
680  IF CL$<"1" OR CL$>"7" THEN 670
```

```
690  LOCATE 0,20 : PRINT "          ";
700  COL=VAL(CL$)
710  COLOR COL
720 RETURN
730 ' --- Gen.End ---
740 *GN.END
750  LOCATE 0,20 : PRINT "GEN.END"
760  AD=&HE000
770  POKE AD,MAX.BY+1 : AD=AD+1
780  POKE AD,MAX.BX+1 : AD=AD+1
790  FOR Y=0 TO MAX.BY
800    FOR X=0 TO MAX.BX
810      GV(0)=PEEK(&HC000+Y*80+X,"0")
820      GV(1)=PEEK(&HC000+Y*80+X,"1")
830      GV(2)=PEEK(&HC000+Y*80+X,"2")
840      FOR I=0 TO 2
850        POKE AD,GV(I): AD=AD+1
860      NEXT I
870    NEXT X
880  NEXT Y
890  CLS
900  SCREEN ,3
910  LOCATE 5,10 : PRINT "Used &HE000 - &H"HEX$(AD-1)
920 'BSAVE "GRPDAT.bin",&HE000,AD-&HE000
930 END
```

7-4-5 高速ROLL文

漢字の出力はグラフィック画面に対して行なわれますので、テキスト画面のように自動的にスクロールするようなことはありません。そこで必要となってくるのがROLL文です。

ROLL文はたいへん便利なコマンドですが、残念ながらN_{88}DISK-BASICでしか使うことができず、また処理速度も速いとはいえません。

そこで、この節では、N_{88}-BASICでの高速ROLL機能を付加するプログラムを紹介します。

リスト 7-9

```
100 '
110 '    SCROLL COMMAND for 640 x 400 dot mode
120 '
130 '    DUMMY = USR(SCROLL.BYT)
140 '
150  DEFINT A-Z
160  RESTORE *ML.DATA
170  SADRS=VARPTR(#0)+9
180  ADRS=SADRS
190 '
200 *WRITE.DATA
210  READ DATUM$
220  IF DATUM$="END" THEN *FINISH
230  POKE ADRS,VAL("&H"+DATUM$)
240  ADRS=ADRS+1
250 GOTO *WRITE.DATA
260 '
270 *FINISH
280  DEF USR=SADRS
290  PRINT "Complete."
300  END
310 '
320 *ML.DATA
```

```
330  DATA 7E,23,66,6F,E5,E5,11,00,C0,19,E3,E5,21,80,3E,C1
340  DATA B7,ED,42,4D,44,E1,E5,D5,C5,F3,D3,5C,ED,B0,D3,5F
350  DATA FB,E1,C1,C5,E5,11,80,FE,7C,F6,C0,67,F3,D3,5D,0A
360  DATA D3,5C,77,D3,5F,FB,23,03,E7,20,F1,C1,D1,E1,C5,F3
370  DATA D3,5D,ED,B0,D3,5F,FB,E1,7C,F6,C0,67,5D,54,1B,C1
380  DATA F3,D3,5D,36,00,ED,B0,D3,5F,FB,C9
390  DATA END
```

上のプログラムを実行すると、あとは

DUMMY＝USR(SCROLL.BYT)

で、グラフィック画面をスクロールさせることができます。 SCROLL.BYTは整数型でなければならず

ROLL n ←→DUMMY＝USR(n＊80)

に対応し、引数を80の倍数以外にすると、斜めにスクロールさせたりすることも可能です。

なお、この機械語プログラムはリロケータブルになっていますので、C000H番地より前であればメモリ上のどこにでも置けます。

7-5 その他のグラフィック用 I／Oポート

7-5-1 CRTのタイプのセンス

PC-8801mkⅡ前面の高解像度CRTと標準CRTの切り換えジャンパースイッチの設定が、どちらの状態になっているかを知ることができます。

ポート 40H (IN)

7	6	5	4	3	2	1	0
／	／	VRTC	CDI	$\overline{EXTON}$	DCD	$\overline{SHG}$	BUSY

SHG: 1：標準CRT / 0：高解像度CRT

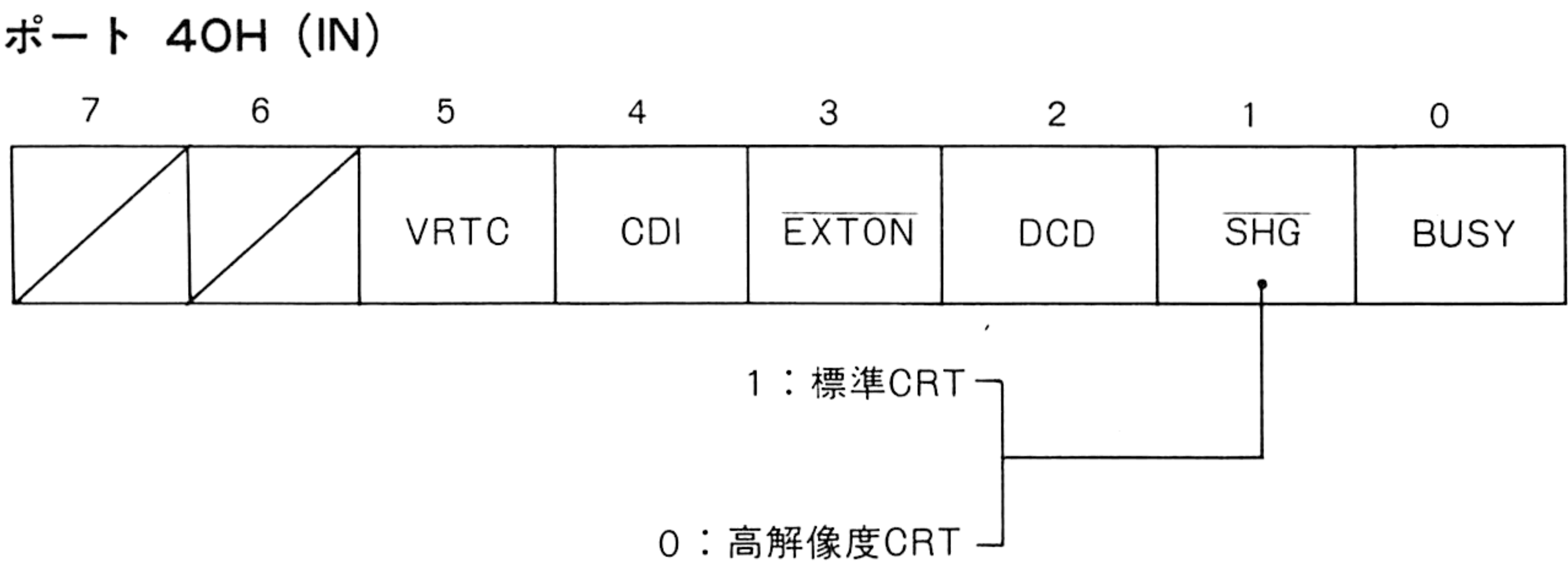



図7-5-A CRTのタイプのセンス

BASICでは次のようにして判別できます。

```
IF INP(&H40) AND 2 THEN 標準CRT
                   ELSE 高解像度CRT
```

7-5-2 バックグラウンドカラー

バックグラウンドカラーとは、グラフィック画面の地の色のことです。

N88-BASICで制御するには、COLOR文を用います。２番目のパラメータがバックグラウンドカラーになるわけですが、白黒モードとカラーモードでは働きが違います。(マニュアルに記述されているのはカラーモードでのことです。)その違いを表にまとめておきます。

バックグラウンドカラーの違い

	白黒モード	カラーモード
パラメータの値	カラーコード	パレット番号
方　　式	ハードウェアによる	ソフトウェアによる

白黒モードでのバックグラウンドカラーはI／Oポート52Hの3bitで制御されます。

ポート 52H（OUT）

7	6	5	4	3	2	1	0
／	BGG	BGR	BGB	／	0	0	0

BGG	BGR	BGB	バックグラウンドカラー
0	0	0	0（黒）
0	0	1	1（青）
0	1	0	2（赤）
0	1	1	3（紫）
1	0	0	4（緑）
1	0	1	5（水色）
1	1	0	6（黄）
1	1	1	7（白）

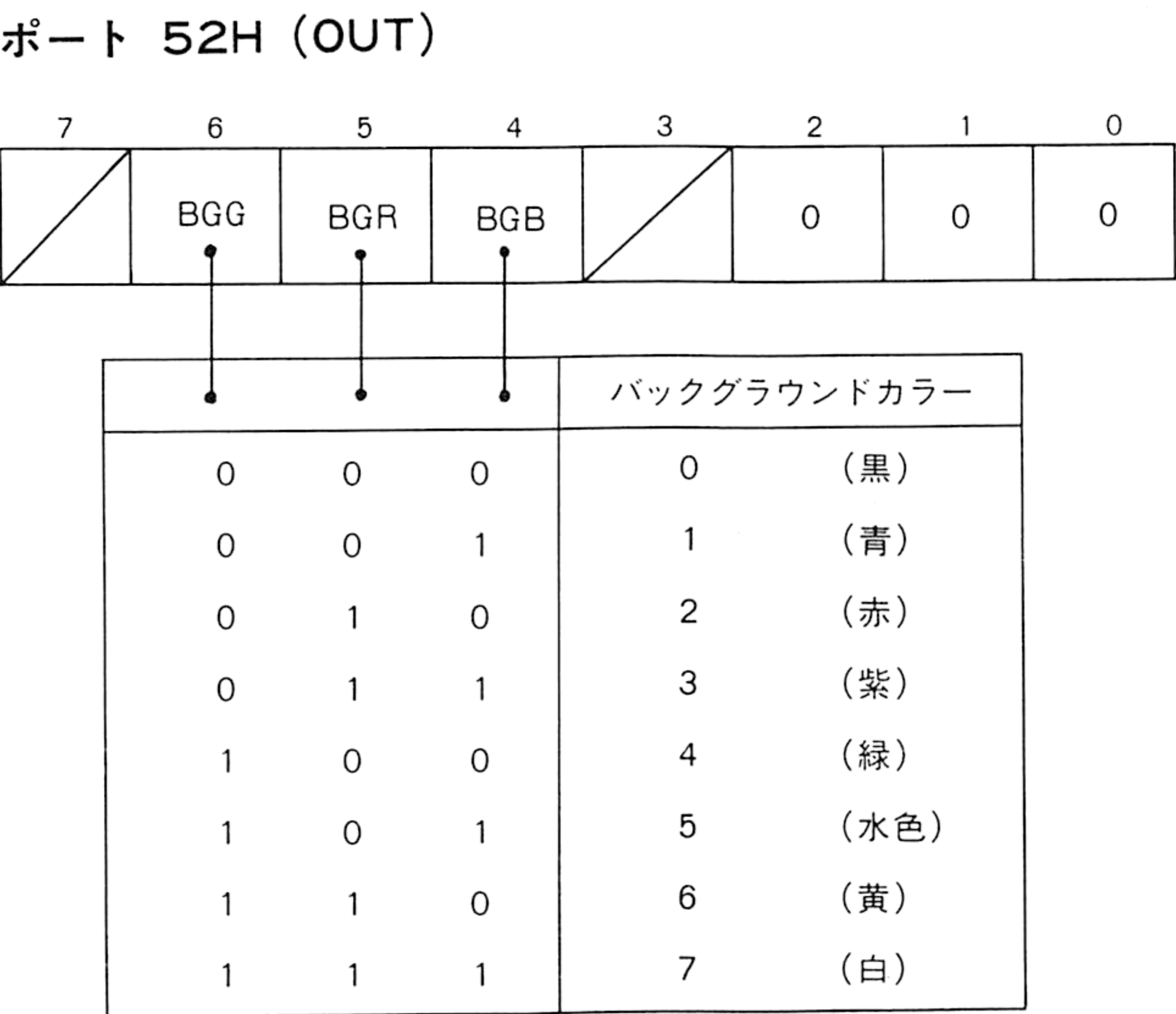



図7-5-B　バックグラウンドカラーの制御

カラーモードでのバックグラウンドカラーは、ソフトウェアによるものです。COLOR文の実行により、ワークエリアF01FH番地にバックグラウンドカラーのパレット番号が書き込まれます。以後、N_{88}-BASICでは、CLS文やPRESET文を実行するときにこの値を参照して、このパレット番号で、画面をクリアしたり点を消したりします。

7-5-3　画面の重ね合わせ

画面のソースにはテキスト、GVRAM0，GVRAM1，GVRAM2とありますが、CRTにどの画面を表示するのかをI／Oポートによって制御できます。ポートアドレスは53Hです。

このポートの使い方は

①テキストモード(ポート31Hのビット３＝０)

グラフィックは使用できませんので、GODS，G1DS，G2DSは意味を持ちません。TXTDSは０にしておきます。TXTDSを１にしますと、CRT上には文字を表示しません。

②モノクログラフィックモード(640×200)

４つの画面ソースとも使用できますので、全てのビットが意味をもちます。

③モノクログラフィックモード(640×400)

GVRAM2は使用しませんので、G2DS以外の３つのビットが意味をもちます。

④カラーグラフィックモード

TXTDSのみ意味をもちます。カラーグラフィックはGODS，G1DS，G2DSにかかわらず表示されます。

ポート 53H（OUT）

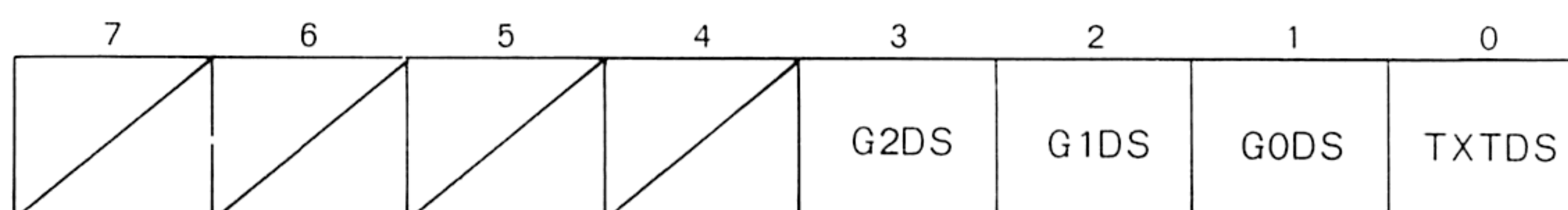
7	6	5	4	3	2	1	0
／	／	／	／	G2DS	G1DS	G0DS	TXTDS

ビット名	画面ソース	0	1
TXTDS	テキスト画面	表示する	表示しない
GODS	GVRAM0	表示する	表示しない
G1DS	GVRAM1	表示する	表示しない
G2DS	GVRAM2	表示する	表示しない

図7-5-C　画面の重ね合わせ制御

7-6　GET文、PUT文

N88-BASICのGET，PUT文はグラフィック画面に表示されているパターンを配列に読み込んだり、配列のデータをグラフィック画面に表示するための命令です。ここでは、GET，PUTで使われるデータの形式や、PUT文の様々な表示方法を見ていきましょう。

7-6-1　GET，PUTのデータ形式

GET，PUTで使われる配列の大きさは、次の式で得られます。

＜添字の値＞＝＜必要なバイト数＞￥N＋＜OPTION　BASE＞＋1…①

＜必要なバイト数＞＝＜横のバイト数＞×＜縦のドット数＞×M＋4…②

N：	整数型配列	N＝2	M：	白黒モード	M＝1
	単精度型配列	N＝4			
	倍精度型配列	N＝8		カラーモード	M＝3

式②の4という数字は、縦横のドット数をそれぞれ2バイトで表わすためのものです。

GET，PUTで使われる配列のデータは、次のような形になっています。

横の ドット数	縦の ドット数	データ 1	データ 2	データ 3	データ 4	データ 5	データ 6	データ 7	データ 8	…… 白黒モード
		データ1B	データ1B	データ1R	データ2R	データ1G	データ2G	データ3B	データ4B	…… カラーモード

8 ドット　8 ドット

5 ドット	データ 1	データ 2	
	データ 3	データ 4	
	データ 5	データ 6	
	データ 7	データ 8	
	データ 9	データ 10	

4 ドット

（12ドット×5ドットのパターン）

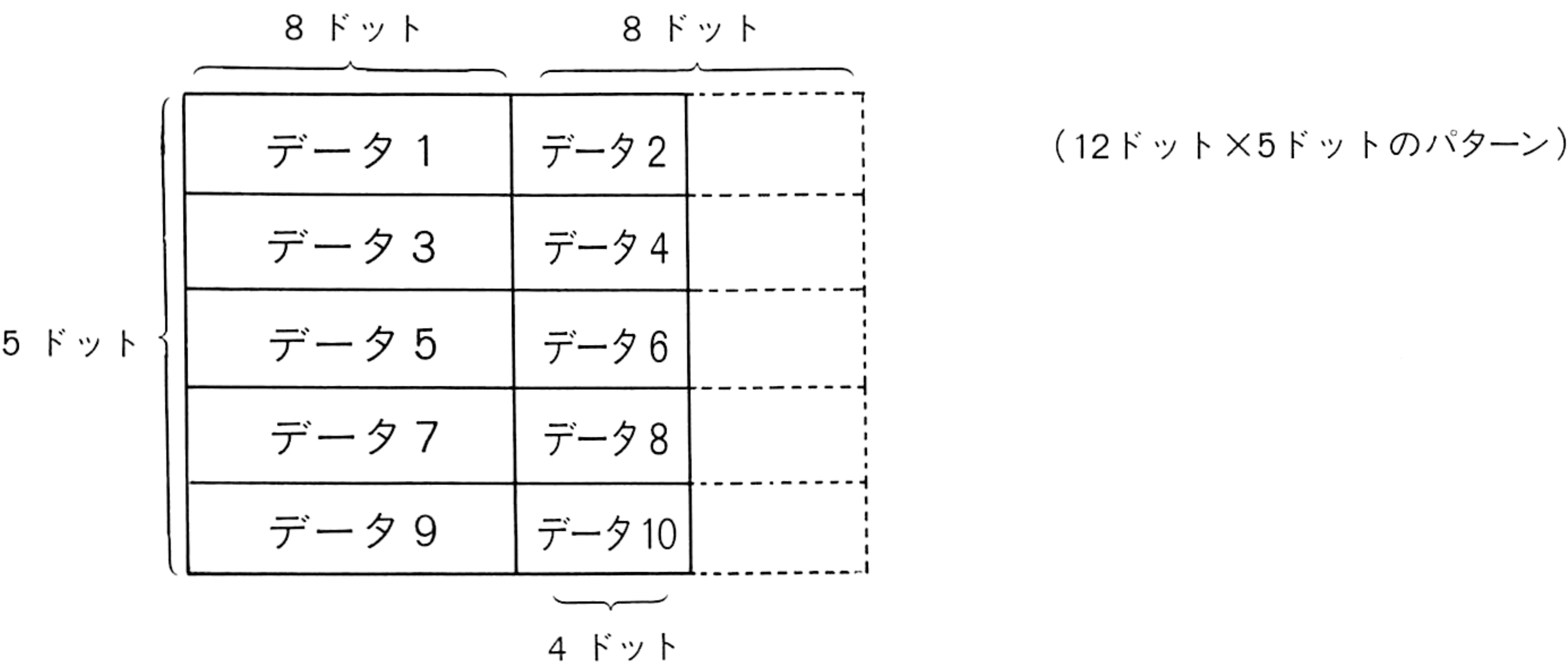



図7-6-A　GET，PUTの配列内データ形式

まず最初に、4バイトのデータが、縦横のドット数を表わします。

次に、グラフィックパターンのデータが、横8ドット、縦1ドットを1つの単位として、白黒モードでは1バイト、カラーモードでは3バイトのデータとなります。

したがって、上の例では、データのバイト数として、白黒モードでは14バイト、カラーモードでは、34バイトが必要になるわけです。

また、このデータと配列の対応は、型や次元数によって次のようになります。

(OPTION BASE 0)

	2バイト	2バイト	1バイト						
	横のドット数	縦のドット数	データ1	データ2	データ3	…	…	…	…
A%(X)	A(0)	A(1)	A(2)		A(3)	A(4)		…	…整数型
A%(2, X)	A (0,0)	A (1,0)	A (2,0)		A (0,1)	A (1,1)		…	…整数型2次元
A!(X)	A(0)		A(1)			…			…単精度型
A#(X)	A(0)					A(1)			…倍精度型

図7-6-B　GET，PUTデータと配列との関係

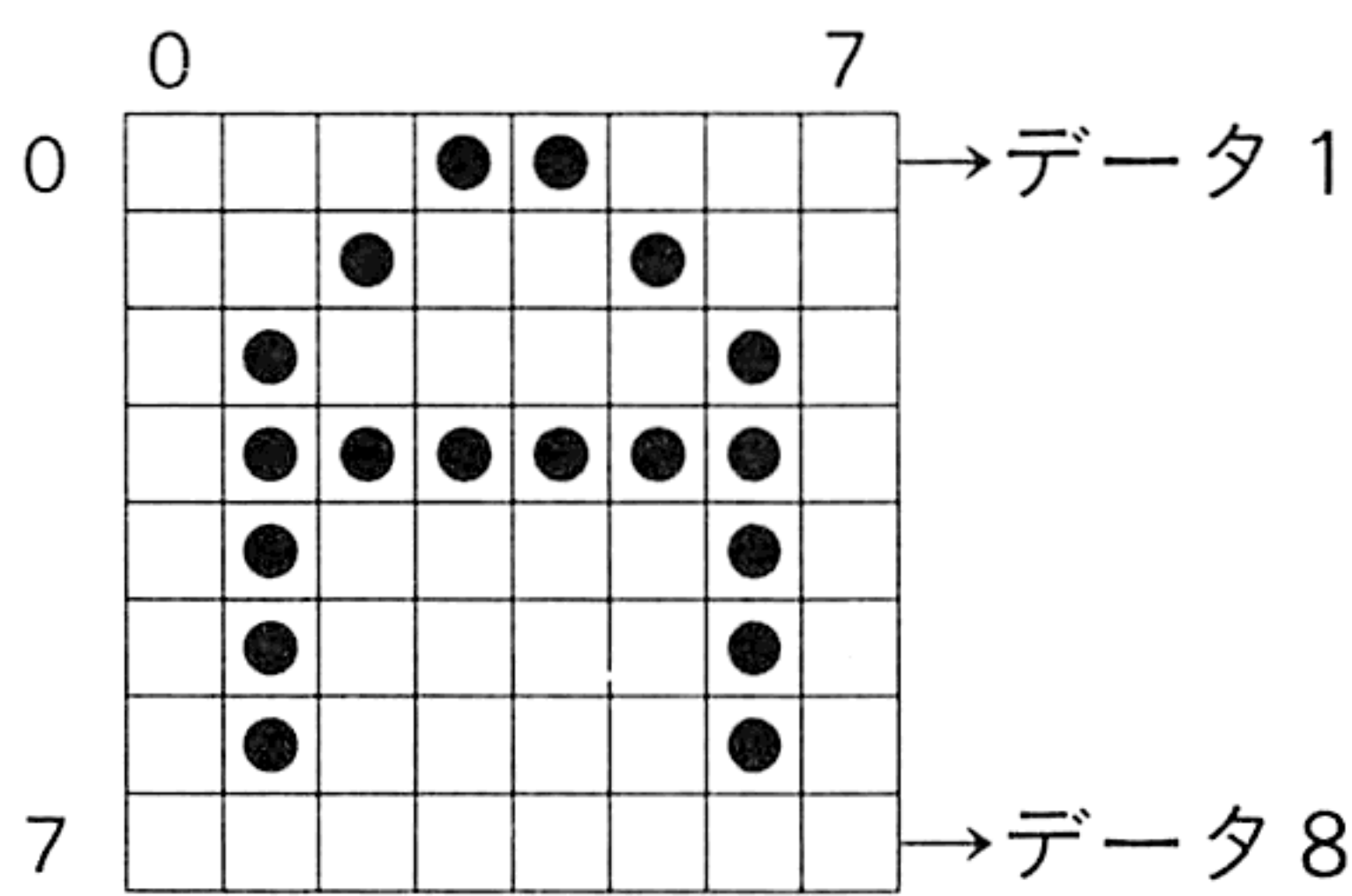
実際の例で見てみましょう。

右のようなパターンが、画面左上に表示されているとします。(640×200ドット、白黒モード) これを整数型配列A%にGETします。

0　　　　　　7
0 →データ1
7 →データ8

図7-6-Cパターン例

リスト　7-10

```
clear
Ok
dim a%(5)
Ok
get(0,0)-(7,7),a%
Ok
```

さて、ここで配列A%の値をPRINTしてもよいのですが、テクノウ的に、配列が格納されているエリアを直接見てみましょう。

リスト　7-11

```
print hex$(varptr(a%(0)))
B2BA
Ok
mon  横のドット数
         縦のドット数
h) db2ba,b2c5         データ1                  データ8
B2BA 08 00 08 00 18 24 42 7E 42 42 42 00
     A%(0) A%(1) A%(2)                 A%(5)
```

7-6-2　複数パターンを1つの配列に

N88-BASICのGET，PUTでは、配列の要素を指定することで、複数のパターンを一つの配列に格納しておくことができます。

これは異なったパターンを表示するときでも、要素の値を変数で持つことによって、1つの文ですますことができる、という利点があります。

ただし、このとき一次元の配列を用いると、要素の計算や大きさを決めるのがめんどうです。そこで複数パターンのための2次元配列の使い方と、その応用例を紹介しましょう。

・配列宣言　(OPTION　BASE　0)

　DIM＜変数名＞(＜最も大きなパターンの添字の最大値＞、＜パターンの数＞-1)

・GET，PUTでの使い方

　＜変数名＞(0，＜パターン番号＞)

　　※GET　(0，0)-(7，7)　PUT　(0，3)など

次の例はグラフィック画面に数字をPUTするサブルーチンです。

数字のデータは配列NDに格納されており、NUMを与えることで、0から9までの数字をPUTすることができるようになっています。

リスト　7-12

```
100 '
110 ' --- PUT number demo ---
120 '
130 SCREEN 0,2 : ROLL 197 : ROLL 3 : SCREEN 1,0
140 DEFINT A-Z
150 DIM ND(5,9)
```

```
160 FOR I=0 TO 9
170   FOR J=0 TO 5
180     READ DA$ : ND(J, I)=VAL("&H"+DA$)
190   NEXT J
200 NEXT I
210 '
220 *NUM. DATA
230 DATA 8,8,423C,5A46,4262,3C
240 DATA 8,8,1808,0828,0808,3E
250 DATA 8,8,423C,0C02,4030,7E
260 DATA 8,8,423C,1C02,4202,3C
270 DATA 8,8,0C04,2414,047E,04
280 DATA 8,8,407E,0478,4402,38
290 DATA 8,8,201C,7C40,4242,3C
300 DATA 8,8,427E,0804,1010,10
310 DATA 8,8,423C,3C42,4242,3C
320 DATA 8,8,423C,3E42,0402,38
330 ' --- test ---
340 FOR NUM=0 TO 9
350   GX=NUM*60+40
360   GY=90
370   GOSUB *NUM. PUT
380 NEXT
390 END
400 ' --- number put sub ----
410 *NUM. PUT
420   PUT(GX,GY),ND(0,NUM)
430 RETURN
```

第8章　漢字ROMと漢字BASIC

PC-8801mkⅡには64Kワード×16ビットの漢字ROMが標準実装されています。これはJIS第1水準の漢字及び非漢字約700種類の文字用のキャラクタジェネレータで、これを使ってグラフィック画面に漢字を表示することができます。N88-BASICではPUT KANJI文を用いて表示しますが、これはたいへん使いにくくかつ遅いので、簡単に漢字を表示できる命令を紹介します。また、漢字出力時にかかせないROLL文の高速版がp. 115(7-4-5 高速ROLL文)にあります。

またPRINT文やREM文中に漢字が使える漢字BASICが発売されています。これを使うと英数字と同じように漢字が出力できます。

8-1　I／Oポート

漢字ROMはI／Oポートを通してアクセスされます。具体的な使い方は後で述べます。

E8H	OUT	漢字ROMアドレス の指定(下位8ビット)	
	IN	漢字フォントデータの読み出し(下位8ビット)	
E9H	OUT	漢字ROMアドレスの指定(上位8ビット)	
	IN	漢字フォントデータの読み出し(上位8ビット)	
EAH	OUT	漢字ROMの読み出し開始	データは何でもよい
EBH	OUT	漢字ROMの読み出し終了	

図8-1-A　漢字ROM　I／Oポート

8-2　漢字フォントのマッピング

漢字フォントのフォーマットには3つの種類があります。

8-2-1　全角文字　(漢字ROMアドレス1200H～FFFFH)

1つの文字データは16×16ドットからなります。1ワード16ビットで16ワード分です。例として漢字ROMアドレス5410H～541FHの文字の「福」を示します。

JISコード4A21H

漢字ROMアドレス

5410H
5411H

541EH
541FH

IN E9H で読み出される　IN E8H で読み出される

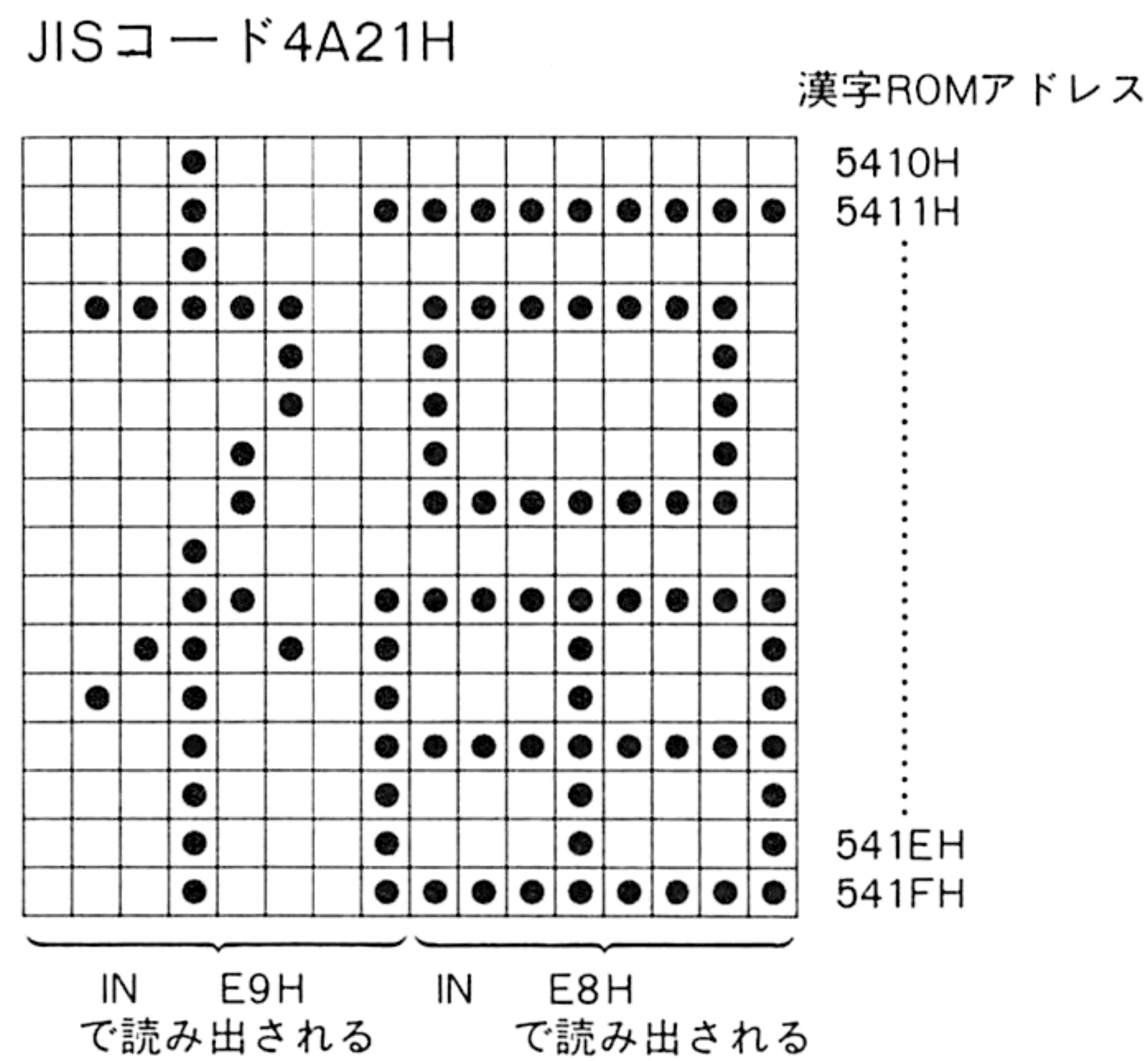



図8-2-A 全角文字のマッピング例

8-2-2 半角文字 (漢字ROMアドレス0000H〜07FFH)

半角文字は8×16ドットで、8ワード(1ワードが2行分)からなります。例として漢字ROMアドレス02C0H〜02C7Hの文字「X」はこうなります。

JISコード0058H

漢字ROMアドレス

HIGH
LOW } 2C0H

HIGH……IN E9Hで読み出される
LOW……IN E8Hで読み出される

HIGH
LOW } 2C7H

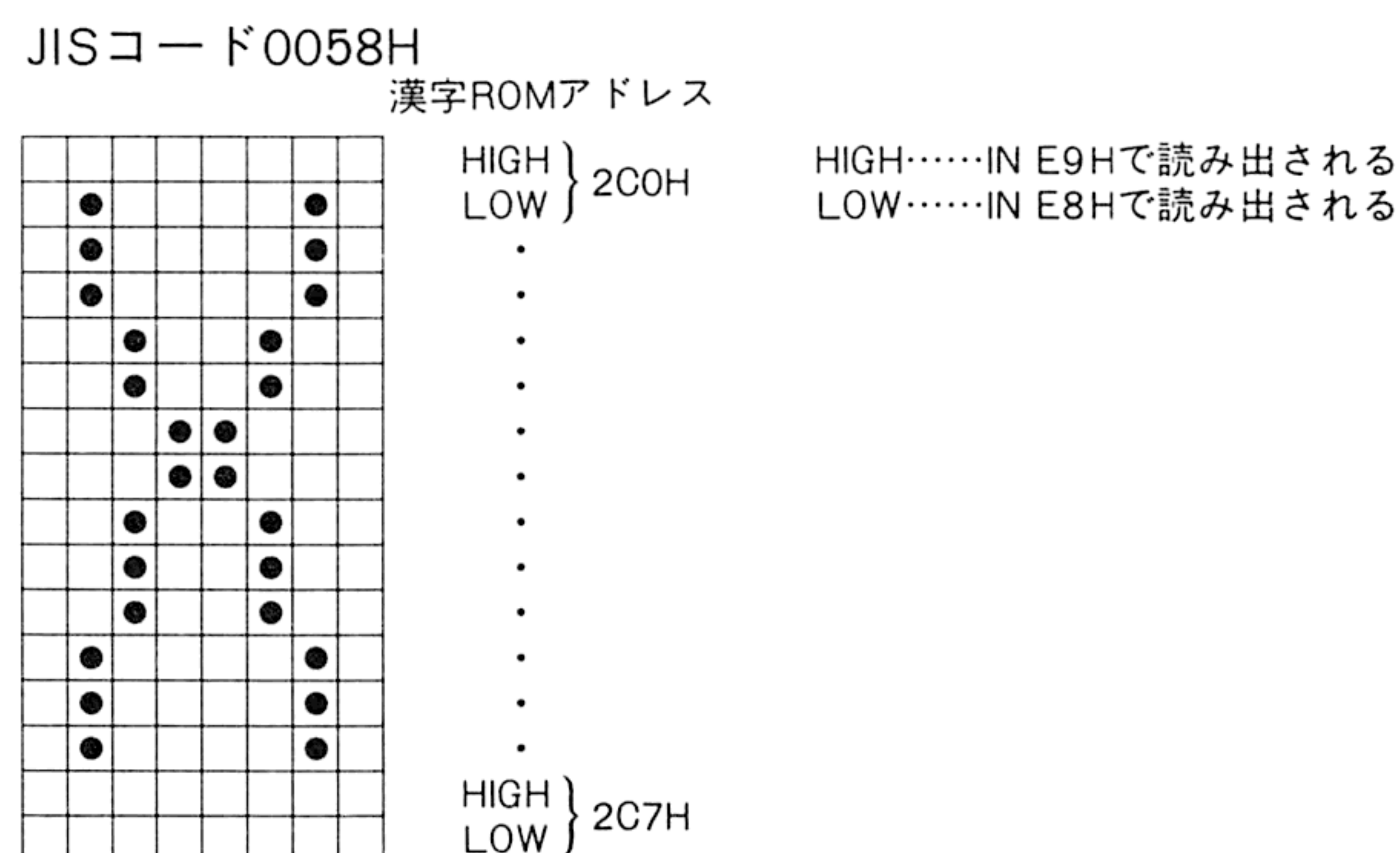



図8-2-B 半角文字のマッピング例

8-2-3 1/4角文字 (漢字ROMアドレス800H〜BFFH)

1/4角文字は、8×8ドットで表示されます。半角文字の上半分と考えればよく、4ワードからなります。例として漢字ROMアドレス0964H〜0967Hの文字「Y」を示します。

JISコード0159H

漢字ROMアドレス

HIGH
LOW } 964H

HIGH……IN E9Hで読み出される
LOW……IN E8Hで読み出される

HIGH
LOW } 967H

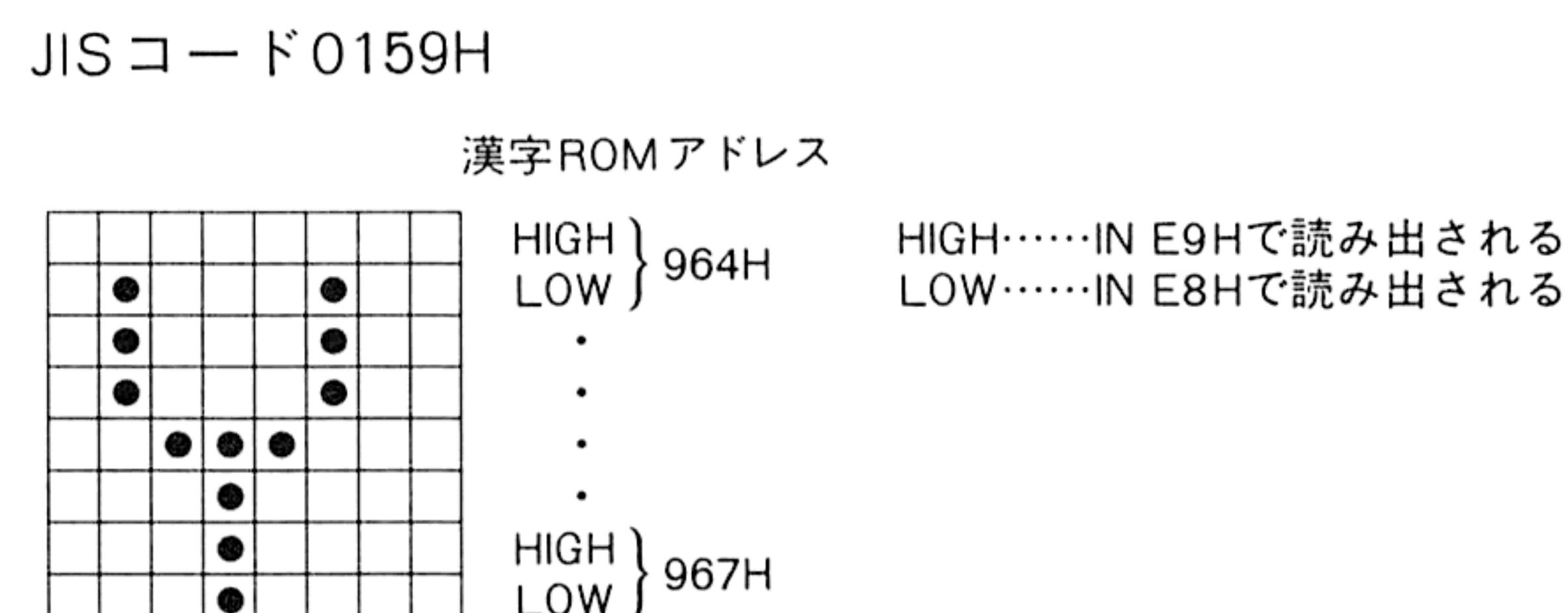



図8-2-C 1/4角文字のマッピング例

8-3　漢字JISコード

漢字ROMアドレスと漢字JISコードの変換方法を述べましょう。漢字JISコードはBASICのPUT KANJI命令で使用するコードで、漢字ROMアドレスは漢字ROMを読むときに使用します。

漢字ROMは、アドレスによって下のように4つに分けられます。

漢字ROMアドレス	デ　ー　タ
0000H～07FFH	半角文字
0800H～11FFH	1/4角文字
1200H～3FFFH	全角文字（非漢字）
4000H～FFFFH	全角文字（漢字）

漢字ROMアドレスとデータの内容

① 半角文字の変換

アドレス　0000H～07FFH　⟷　JISコード　0020H～00FFH

（例）

漢字JISコード　| 0 0 0 0 0 0 0 0 | × × × × × × × × |　0058H(半角文字“X”)

15　　　0

↕

漢字ROMアドレス　| 0 0 0 0 0 | × × × | × × × × × | △ △ △ |　02C0H～02C7H

15

△△△：$000_{(2)}$～$111_{(2)}$（0～7）

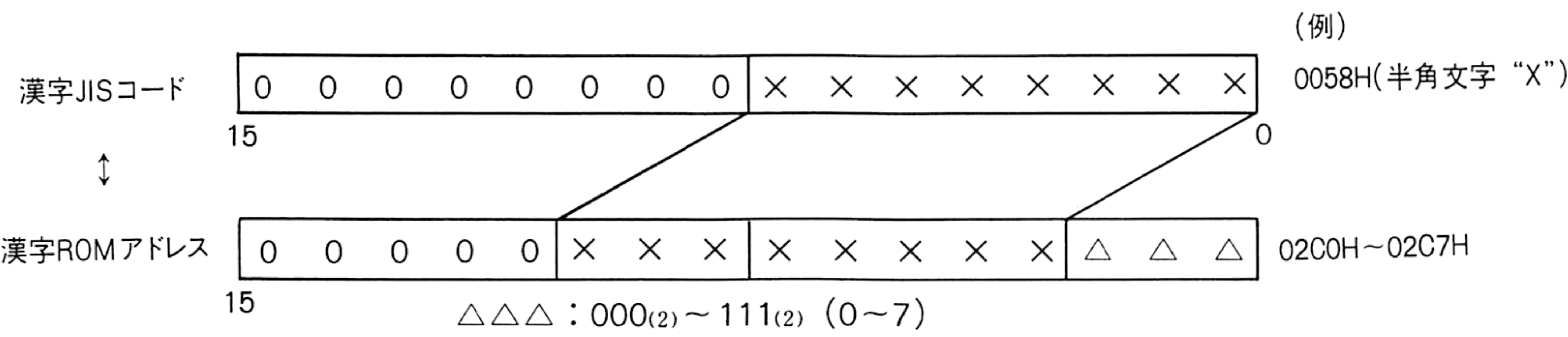



図8-3-A　半角文字の変換

② 1／4角文字の変換

アドレス　0800H～0BFFH　⟷　JISコード　0100H～01FFH

（例）

漢字JISコード　| 0 0 0 0 0 0 0 1 | × × × × × × × × |　0159H(¼角文字“Y”)

15　　　0

↕

漢字ROMアドレス　| 0 0 0 0 1 0 | × × × × × × × × | △ △ |　0964H～0967H

15　　　0

△△：$00_{(2)}$～$11_{(2)}$（0～3）

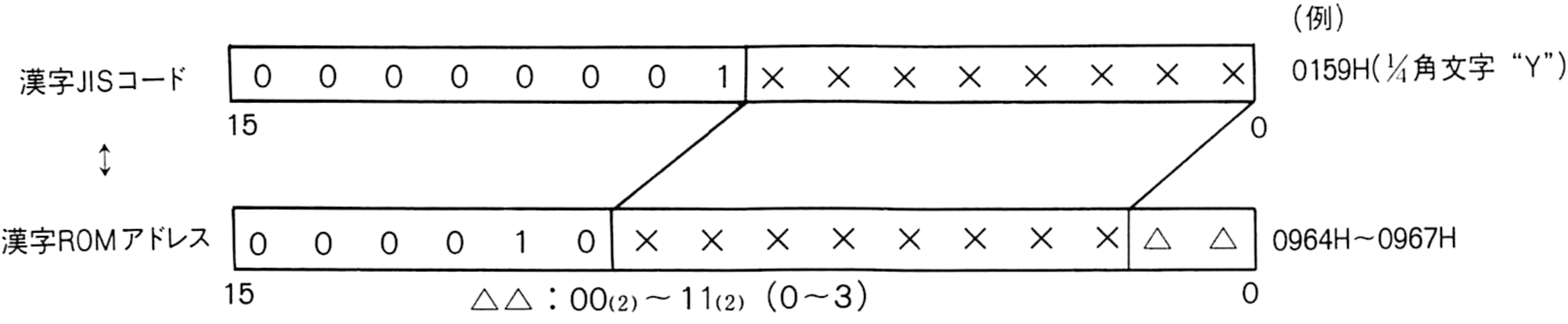



図8-3-B　1／4角文字の変換

③　全角文字(非漢字)の変換

アドレス　1200H～3FFFH　⟷　JISコード　2120H～277FH

(例)

	15															0	
漢字JISコード	0	0	1	0	0	×	×	×	0	×	×	×	×	×	×	×	235AH(英字 “Z”)
漢字ROMアドレス	0	0	×	×	×	×	×	×	×	×	×	×	△	△	△	△	27A0H～27AFH

△△△△：$0000_{(2)}$～$1111_{(2)}$ (0～15)

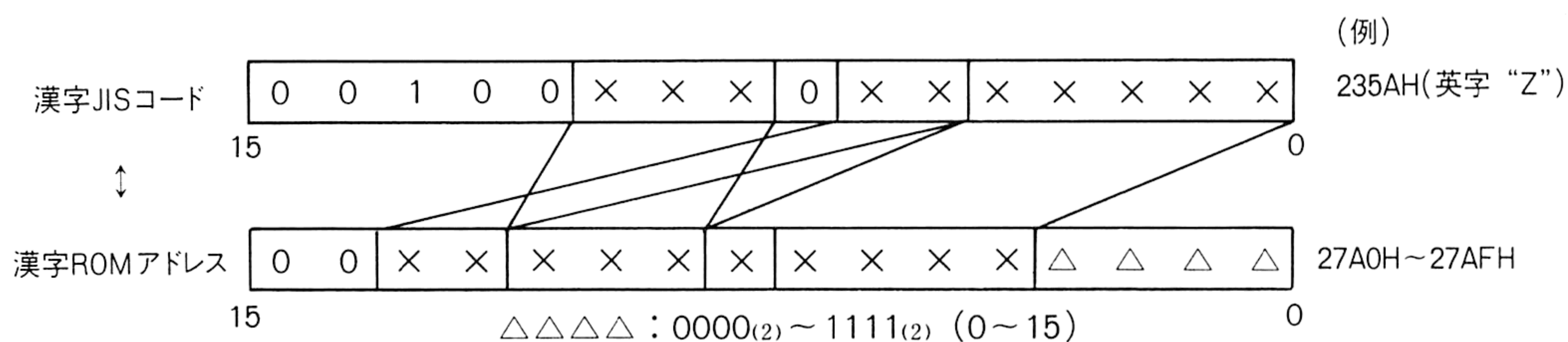



図8-3-C　全角文字(非漢字)の変換

④　全角文字(漢字)の変換

アドレス　4000H～FFFFH　⟷　JISコード　3020H～4F5FH

(例)

	15															0	
漢字JISコード	0	× (A)	× (B)	×	×	×	×	×	0	×	×	×	×	×	×	×	4A21H(漢字 “福”)
漢字ROMアドレス	×	×	× (C)	×	×	×	×	×	×	×	×	×	△	△	△	△	5410H～541FH

△△△△：$0000_{(2)}$～$1111_{(2)}$ (0～15)

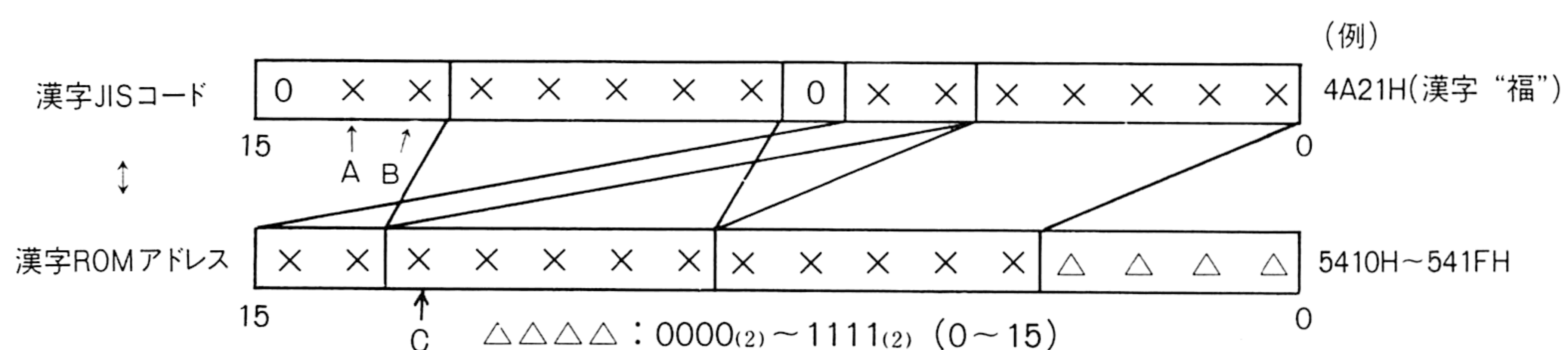



図8-3-D　全角文字(漢字)の変換

漢字ROMアドレスからJISコードを求める場合、アドレスのビット13(C)によってJISコードのビット14，13(A，B)を決めます。

C＝0のとき　　　AB＝10

C＝1のとき　　　AB＝01

と割り当てます。

このように、漢字JISコードから漢字ROMアドレスに変換するのは面倒な作業です。そこでBASICで書いた変換プログラムをあげておきます。

リスト　8-1

```
100 '
110 '     KANJI JIS Code => KANJI ROM Address
120 '
130 *INP.KCODE
140  INPUT "KANJI JIS Code ? &H",KC$
150  K.COD=VAL("&H"+KC$)
160   IF K.COD>=  &H20 AND K.COD<=  &HFF THEN *H.CHR
170   IF K.COD>= &H100 AND K.COD<= &H1FF THEN *F.CHR
180   IF K.COD>=&H2120 AND K.COD<=&H277F THEN *N.KNJ
190   IF K.COD>=&H3020 AND K.COD<=&H4F5F THEN *KNJ
200 GOTO *INP.KCODE
210 '
220 *H.CHR
230  K.ADR=K.COD*8
```

```
240  K.BYT=8
250 GOTO *PRN.ADR
260 '
270 *F.CHR
280  K.ADR=(K.COD-&H100)*4+&H800
290  K.BYT=4
300 GOTO *PRN.ADR
310 '
320 *N.KNJ
330  K1=(K.COD AND &H60)*&H80
340  K2=(K.COD AND &H700)*2
350  K3=(K.COD AND &H1F)*&H10
360  K.ADR=K1+K2+K3
370  K.BYT=16
380 GOTO *PRN.ADR
390 '
400 *KNJ
410  K1=(K.COD AND &H60)*&H200
420  K2=(K.COD AND &H1F00)*2
430  K3=(K.COD AND &H1F)*&H10
440  K.ADR=K1+K2+K3
450  K.BYT=16
460 '
470 *PRN.ADR
480  PRINT "--------------"
490  PRINT "KANJI JIS CODE : &H";HEX$(K.COD)
500  PRINT "KANJI ROM ADRS : &H";HEX$(K.ADR)" - &H";HEX$(K.ADR+K.BYT-1)
510  PRINT "--------------"
520 '
530 GOTO *INP.KCODE
```

8-4 漢字ROMからのデータの読み出し方

漢字ROMからのデータの読み出しには、I/Oポートによって、漢字ROMをアクセスします。普通はPUT KANJI命令で表示しますから漢字ROMを読む必要はないのですが、機械語で漢字を出力したり、漢字を大きく表示したりする場合などには漢字ROMから漢字フォントを読み出さなくてはなりません。

漢字ROMの１ワードを読み出すアルゴリズムは次のようになります。

① 漢字ROMアドレスの下位８ビットをセットします。(OUT 0E8H)

② 漢字ROMアドレスの上位８ビットをセットします。(OUT 0E9H)

③ ROMへのアクセスを開始します。データは何でもかまいません。(OUT 0EAH)

④ 時間待ちをします。(NOP２回程度。BASICでは必要ありません。)

⑤ フォント右側の８ビットデータを読み出します。(IN 0E8H)

⑥ フォント左側の８ビットデータを読み出します。(IN 0E9H)

⑦ ROMへのアクセスを終了します。データは何でもかまいません。(OUT 0EBH)

①と②は順番はどちらでもかまいません。⑤と⑥も同様です。また、漢字ROMをアクセスするプログラムでは、初めにOUT 0EBH(漢字ROMのアクセス終了命令)を行なっておいた方がよいでしょう。

8-4-1 BASICで

まずは、先のアルゴリズムに従って、BASICによる漢字フォントデータ読み出しプログラムを作ってみます。

このプログラムは、漢字JISコードを入力するとそのデータを読み出し、画面上に表示するというものです。漢字コードは、3021H～4F53Hと制限がついていますが、半角文字、1/4角文字についても同じような方式でデータを読み出すことが出来ます。

リスト 8-2

```
100 '
110 '    Read KANJI Character Font
120 '
130 DEFINT A-Z
140 WIDTH 40,25
150 '
160 INPUT "KANJI Code (&H3021 - &H4F53)";K.CODE
170 IF K.CODE<&H3021 OR K.CODE>&H4F53 THEN 160
180 '
190 GOSUB *KC.TO.KA
200 '
210 FOR KA=K.ADRS TO K.ADRS+15
220   KA.L=PEEK(VARPTR(KA))
230   KA.H=PEEK(VARPTR(KA)+1)
240   OUT &HE8,KA.L
250   OUT &HE9,KA.H
260   OUT &HEA,0
270   L.DAT=INP(&HE9)
280   R.DAT=INP(&HE8)
290   OUT &HEB,0
300   GOSUB *PRINT.PAT
310 NEXT KA
320 '
330 END
340 '
350 *KC.TO.KA    :' Get KANJI-ROM Address from KANJI Code
360  KA1=(K.CODE AND &H60)*&H2
370  KA2=(K.CODE AND &H1F00)*2
380  KA3=(K.CODE AND &H1F)*&H10
390  K.ADRS=KA2+KA3
395  POKE VARPTR(K.ADRS)+1,PEEK(VARPTR(K.ADRS)+1) OR KA1
400 RETURN
410 '
420 *PRINT.PAT
430  PRINT HEX$(KA)"H : ";
440  DAT=L.DAT : GOSUB *PRINT.DOT
450  DAT=R.DAT : GOSUB *PRINT.DOT
460  PRINT
470 RETURN
480 *PRINT.DOT
490  FOR B=7 TO 0 STEP -1
500    IF DAT AND (2^B) THEN PRINT "■"; ELSE PRINT "  ";
510  NEXT B
520 RETURN
```

8-4-2　機械語で(ROM内ルーチンを使って)

N88-BASICでは、漢字を出力することができるわけですから、そのためのルーチンがROM内にあるはずです。それを使わない手はないというので、ここではROMルーチンを使って漢字ROMのデータを読み出してみましょう。

このルーチンは、ROM 5 に納められており、次のように使います。

START
DE←漢字JISコード
ROM3をアクティブにする
CALL 7261H
メインROMをアクティブにする
END

※実行後データは次の形でワークエリア上に格納されている。

E9B9H [ ] [00] [ ] [00] [データ]

横のドット数(8または10H)
縦のドット数(8または10H)
漢字のデータ(8, 16, 32バイト)

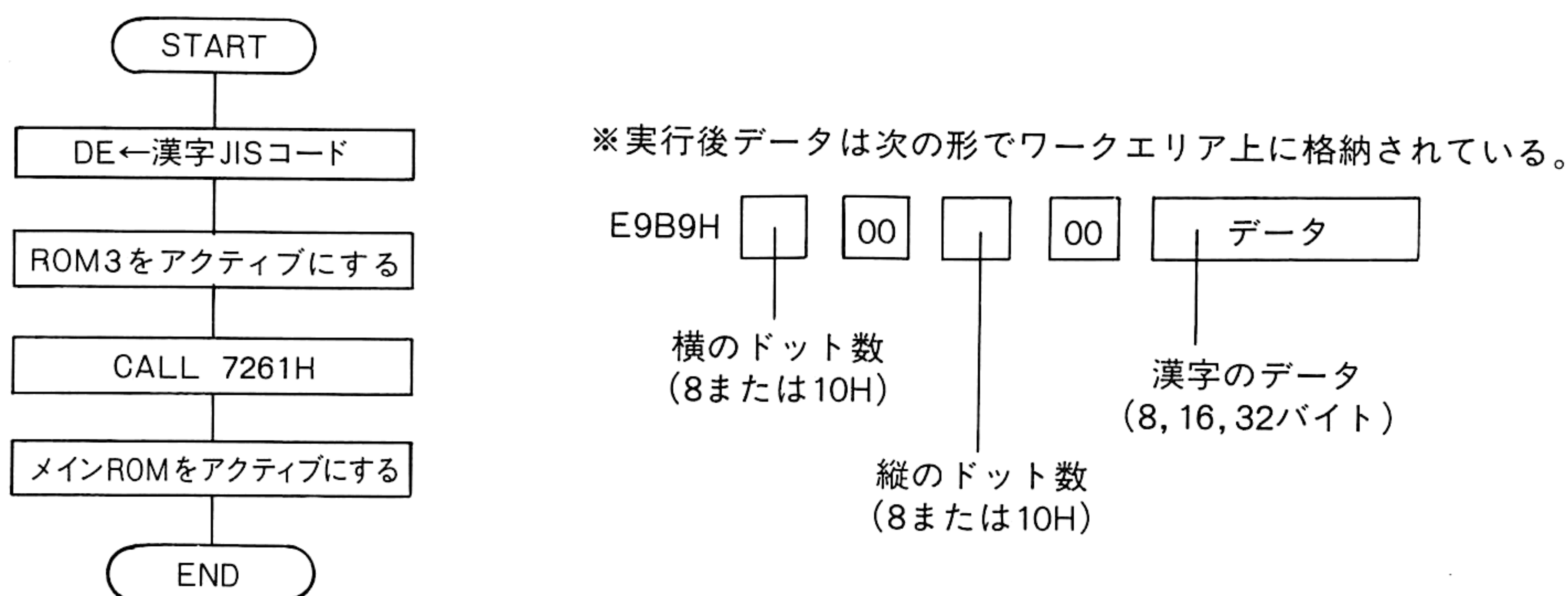



図8-5-A　漢字ROMデータ読み出しルーチン

実は、このルーチンは、PUT KANJIの一部で、余分な処理まで行なってしまいますが、実用上は問題ありません。なお、このワークエリア(E9B9H～)は、行入力バッファも兼ねていますので、ROMルーチンから戻った後、INPUTなどを実行すると、読み出したデータは消されてしまいますから、読み出した後は別のところへ移しておく方が安全です。

次のプログラムは、漢字コードを入力すると、そのデータを16進で出力するというもので、半角文字、1/4角文字でも使えます。

リスト　8-3

```
100 '
110 '   Read KANJI Character Font
120 '                    ( Using N88-BASIC ROM Routine )
130 '
140 DEFINT A-Z
150 '
160 DEF USR=&HF2E0
170 FOR I=&HF2E0 TO &HF2F2
180   READ D$ : POKE I,VAL("&H"+D$)
190 NEXT
200 DATA 7E,23,66,6f,EB,F3,3E,FE,D3,71,CD,61,72,3E,FF,D3
210 DATA 71,FB,C9
220 '
230 DT.TOP=&HE9B9
240 '
250 INPUT "KANJI JIS Code ? &H",KC$
260 KC=VAL("&H"+KC$)
270 '
280 DUM=USR(KC)
290 '
300 PX=PEEK(DT.TOP) : PY=PEEK(DT.TOP+2)
310 DT.TOP=DT.TOP+3
320 FOR I=1 TO (PX/8)*PY
330  PRINT RIGHT$("0"+HEX$(PEEK(DT.TOP+I)),2)" ";
340 NEXT I
```

8-5 高速漢字PUT文

この章の初めで述べたとおり、漢字をPUT KANJIで一文字ずつ出していくのはたいへんです。漢字BASICを使えばPRINT文中に漢字を使えるので一番よいのですが、そこまでいかなくてもJISコード列を文字列として持って、それを一度にPUTできればだいぶ楽になります。

ここでは

POLL ＜JISコード文字列＞

で全角の文字列を任意の場所(ただし横方向は8ドットごと)にPUTできるようなプログラムを紹介します。

＜JISコード文字列＞というのは漢字JISコードを上位、下位の順にならべていったもので、たとえば「福岡市」という漢字列は

CHR$(&H4A)+CHR$(&H21)+CHR$(&H32)+CHR$(&H2C)+CHR$(&H3B)+CHR$(&H54)
福　　　　　　　　　　岡　　　　　　　　　　市

ということになりますが、これではあんまりなので「第11章　プリンタ」で紹介する漢字キャラクタ対応表(完全な表は付録にあります)を使うと

”J！2，;T”

となり、あいかわらずわけがわかりませんが、たいへん短く扱いやすくなります。

つぎのプログラムを実行してください。漢字列をPUTするPOLL文ができあがります。

リスト 8-4

```
100 '
110 '  KANJI STRING PUT
120 '
130 DEFINT A-Z
140 FOR AD=&H866A TO &H8729
150   READ D$ : POKE AD,VAL("&H"+D$)
160 NEXT
170 POKE &HEEA7,&HC3 : POKE &HEEA8,&H6A : POKE &HEEA9,&H86
180 END
190 '
200 ' 866A-86E9
210 DATA 00,CD,D3,11,E5,CD,C9,56,7E,F5,23,5E,23,56,D5,D3
220 DATA EB,CD,EB,86,E5,DD,E1,E1,F1,FE,02,38,10,3D,3D,F5
230 DATA 7E,E6,7F,57,23,7E,E6,7F,5F,23,E5,18,02,E1,C9,EB
240 DATA 7C,FE,30,30,0C,87,E6,0E,57,7D,0F,E6,30,B2,57,18
250 DATA 0A,E6,1F,87,57,7D,E6,60,87,B2,57,7D,87,87,87,87
260 DATA 5F,7A,CE,00,57,06,10,DD,E5,E1,C5,7B,D3,E8,7A,D3
270 DATA E9,13,D3,EA,00,00,DB,E9,CD,0E,87,23,DB,E8,CD,0E
280 DATA 87,D3,EB,01,4F,00,09,C1,10,E0,DD,23,DD,23,C3,81
290 ' 86EA-8729
300 DATA 86,2A,29,F0,29,29,29,44,4D,29,29,09,29,44,4D,2A
310 DATA 27,F0,CB,3C,CB,1D,CB,3C,CB,1D,CB,3C,CB,1D,09,01
320 DATA 00,C0,09,C9,D5,57,0E,5C,3A,1E,F0,06,03,F3,ED,79
330 DATA 0F,30,03,72,18,02,36,00,0C,10,F3,D3,5F,FB,D1,C9
```

このPOLL文はカラーグラフィックモードで動作し、漢字列をPUTする位置はLP(Last Referenced Point)で指定します。つまり、(80,100)に表示したいときはPOINT(80,100)と行なってからPOLL文を実行します。X方向の位置は8の倍数でなければなりません。なお、改行やスクロールは行なわれませんし、LPの範囲チェックも行なっていませんので注意してください。

LP
↓
漢　字　列
16ドット
16×nドット

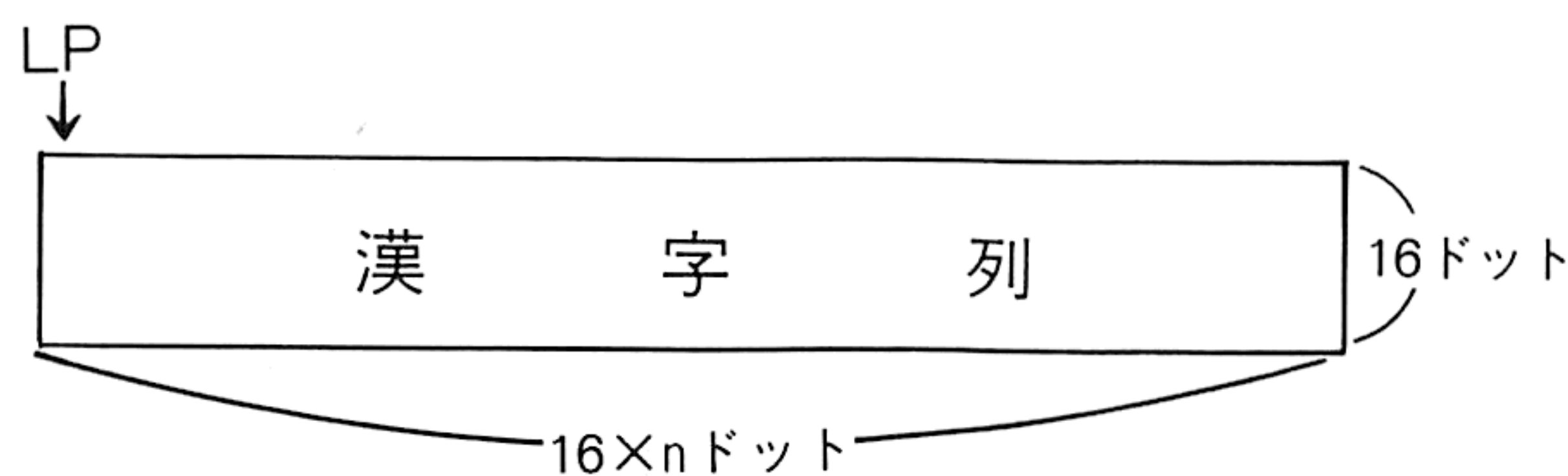



図8-6-A　漢字列の表示位置

例として、(80、100)にパレット6で「福岡市」とPUTするには

```
10  POINT (80, 100)
20  COLOR  ,,,  6
30  POLL  "J!2, ;T"
```

とします。バックグラウンドカラーは使用できません。

8-6　漢字BASIC

PC-8801mkⅡ用の漢字BASICとしては、NECから販売されている「N88-漢字BASIC」の他に、システムソフトから販売(東海クリエイト開発)されている「8801漢字BASIC」があります。どちらも機能としては同じようなもの(単漢字変換)ですが、ここでは表示が高速な8801漢字BASICを例にとって説明します。この漢字BASICの特徴は次のとおりです。

①PRINT文、DATA文、REM文、その他文字列の中に自由に日本語が組み込める。
②必要な部分だけをグラフィック表示するために表示が高速。
③1／4角が表示できる。(プリンタには出力できない)
④すべてオンメモリで処理するため、辞書ファイルのアクセスがなく、すべてのドライブが自由に使える。
⑤94文字の外字が使える。
⑥漢字処理のためのエリアの減少は、テキストRAMの9Kバイトだけですむ。
⑦メモリマップがN88DISK-BASICとほぼ同じであるため、N88DISK-BASIC用の多くの機械語プログラムが使える。拡張命令パッケージ(CMDシリーズ)も動くようです。(メーカーは保証していませんが。)
⑧入力時に書式制御ができる。事務処理プログラムで重宝します。

問題点としては、

①専用高解像度ディスプレイでないと使えない。
②SCREEN　0と1が使えない。
③REM文の代わりに'(アポストロフィー)を使うとそこでは一部の漢字が使えない。
④漢字に新しい読みをつけられない。

⑤外字で24×24ドットのものが作れない。(24ピンプリンタ出力時に問題となる。)
があります。

8-6-1　メモリマップ

FFFF
メインRAM
ディスクコード
8000
7FFF
N88-BASIC
ROM
0000

C000
GVRAM 0
表示

GVRAM 1
表示

GVRAM 2
ワーク

メインルーチン
5C00
テキスト
RAM

ROM 3

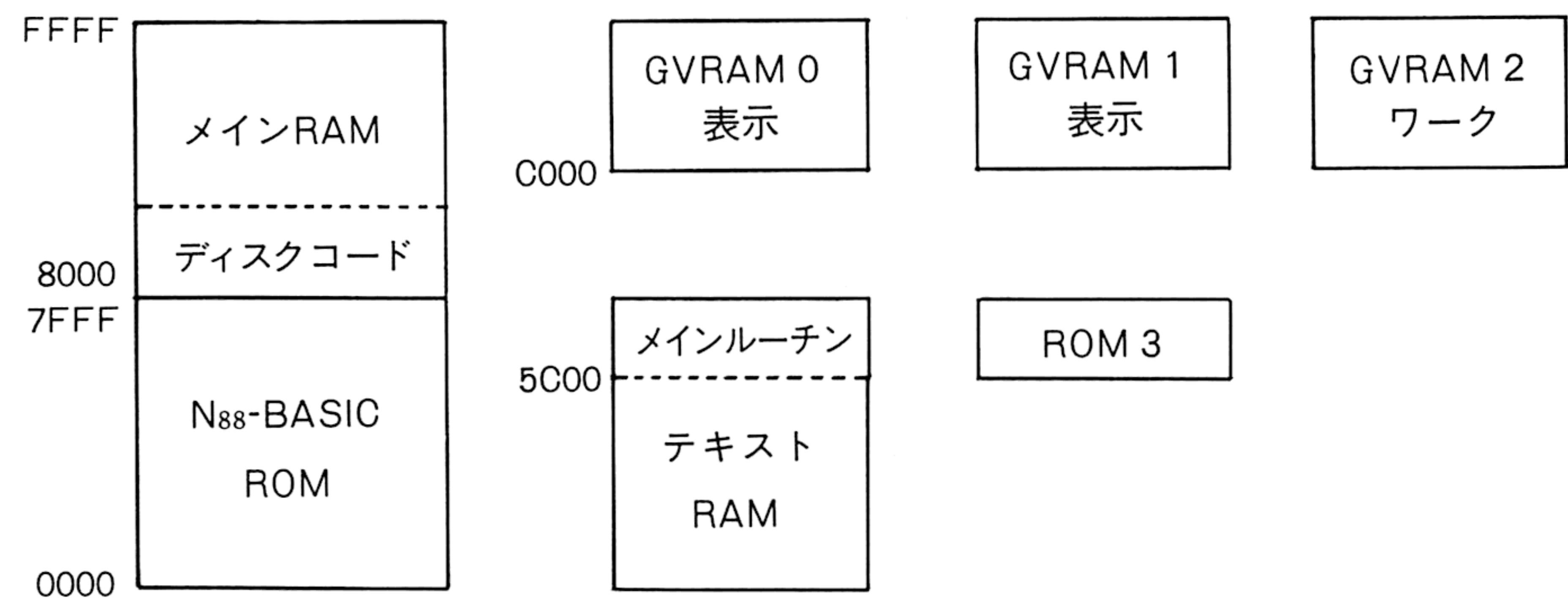



図 8-6-A　8801漢字BASICメモリマップ

このようにテキストRAMは9Kバイト減少しますが、メインRAMのマップはN88DISK-BASICとほぼ同じです。

8-6-2　日本語文字列の内部表現

日本語文字列は1文字が2バイトで表わされ、文字列の両端を7FHで囲んだ形で格納されています。日本語文字のコードはJISコードを基本としていますが、少し変更されています。

①全角文字

JISコードの上位、下位バイトそれぞれの最上位ビットを1にしたものを上位、下位の順に格納されています。

②半角文字と1/4角文字

半角文字のJISコードは0020H～00FFHで、1/4角文字は0100H～01FFHです。どちらも同じく下図のように変換されたあと、上位、下位の順に格納されています。

JISコード　0 0 0 0 0 0 0 | × × × | × × × × × ×
15　0

内部コード　1 0 0 0 0 | × × × | 1 0 | × × × × × ×
15　0

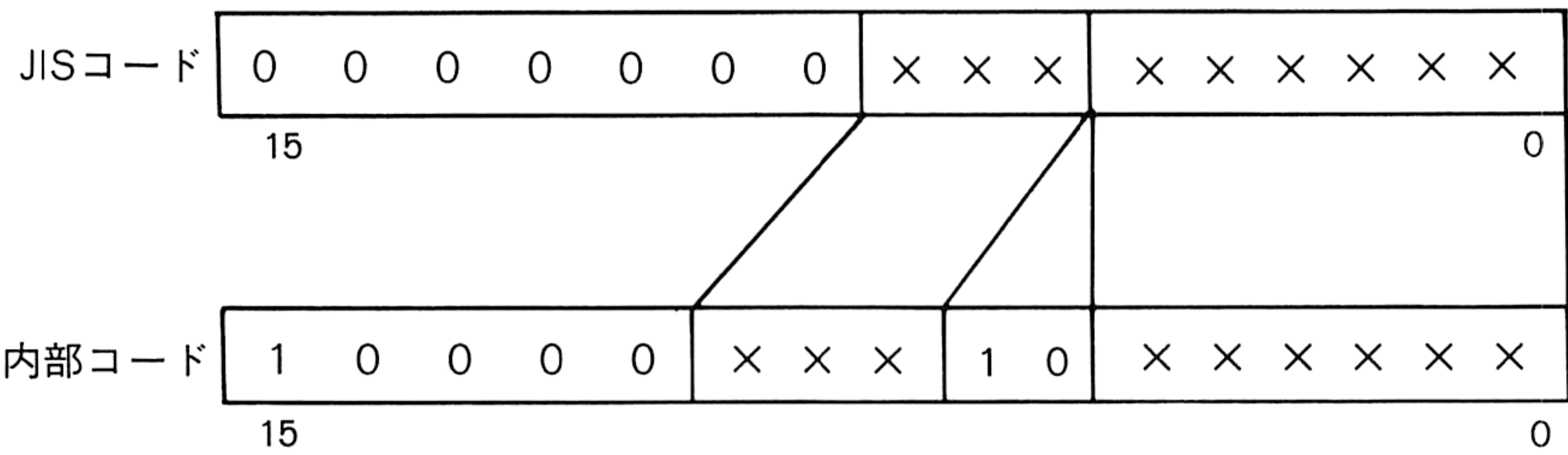



図 8-6-B　半角・1／4角文字の内部コードへの変換

たとえば”日本語表示JISコード″”という日本語文字列は

7F C6FC CBDC B8EC C9BD BCA8 818A 8189 8193 82BA 82B0 8384 839E 7F

KIN 日 本 語 表 示 J I S コ ー ド ″ KOUT

となります。

もう少し詳しくみてみましょう。普通の文字列と日本語文字列が混ざったときには次のようになります。

41 42 7F B4 C1 BB FA 7F 43 44

A B KIN 漢 字 KOUT C D

また、日本語文字列どうしを結合させたときは、間の７Ｆは自動的に取り除かれます。

７Ｆ福岡市７Ｆ＋７Ｆ中央区７Ｆ → ７Ｆ福岡市中央区７Ｆ

（８バイト） （８バイト） （14バイト）

したがって文字列が日本語を含む場合、LEN(A$＋B$)がLEN(A$)＋LEN(B$)よりも小さくなる場合があります。

8-7-3 日本語文字列の処理

このように日本語文字列は普通の文字列を扱う場合に比べて複雑になっていますし、日本語もJISコードそのものが入っているわけではありませんから、その変換も必要です。

そこでBASICで処理する方法の例をいくつかあげてみます。

①JISコードを日本語文字列に変換する。

整数型変数JISにはいっているJISコードを１文字の日本語文字列MJ$に変換するサブルーチンです。変数JISには正しいJISコードが入っていなければなりません。

リスト 8-5

```
100 *JIS.TO.MOJI
110  ZP%=VARPTR(JIS%) : MJ$=CHR$(&H7F)
120  IF JIS%<&H2020 THEN 150
130    MJ$=MJ$+CHR$(PEEK(ZP%+1) OR &H80)+CHR$(PEEK(ZP%) OR &H80)+MJ$
140    RETURN
150  IF JIS%<&H20 OR JIS%>=&H200 THEN 190
160    MJ$=MJ$+CHR$((PEEK(ZP%+1)*4+PEEK(ZP%)¥&H40) OR &H80)
170    MJ$=MJ$+CHR$((PEEK(ZP%) AND &H3F) OR &H80)+CHR$(&H7F)
180    RETURN
190 MJ$="" : RETURN
```

②日本語文字列をJISコードに変換する。

今度は逆に、日本語文字列MJ$の最初の1文字をJISコードに変換して、整数型変数JISに入れます。MJ$は正しい日本語文字列でなければなりません。

リスト 8-6

```
100 *MOJI.TO.JIS
110  JIS%=0 : ZP%=VARPTR(JIS%)
120  IF ASC(MJ$)<>&H7F OR LEN(MJ$)<4 THEN RETURN
130  POKE ZP%,ASC(MID$(MJ$,3,1)) AND &H7F
140  POKE ZP%+1,ASC(MID$(MJ$,2,1)) AND &H7F
150  IF JIS%>&H2020 THEN RETURN
160  POKE ZP%,((PEEK(ZP%+1)AND 3)*&H40) OR PEEK(ZP%)
170  POKE ZP%+1,PEEK(ZP%+1)¥4
180 RETURN
```

③普通の文字(１バイト系)を日本語文字(２バイト系)に変換する。

これは処理速度の点からいっても、表を用意するのがよさそうです。つまり、１バイト系文字256種に対して、それぞれに対応する２バイト系文字を用意しておいて、それをみながら変換するという方法です。BASICで行なう場合、最も簡単なのは、256の大きさを持った文字型配列を用意し、あらかじめ各要素にキャラクタコード00H～FFHに対応する日本語文字を入れておいて、変換したい１バイト系文字のキャラクタコードを添字として使って、配列から日本語文字を取り出すという方法です。

④日本語文字(２バイト系)を普通の文字(１バイト系)に変換する

これも③と同様な表を用意して、その中から変換したい２バイト系文字を捜し出し、それが何番目の要素であるかを数え、それをCHR$()にかければ、１バイト系の文字が得られます。

これにはSEARCH文が最適です。SEARCH文では整数型配列しか使えませんが、日本語文字の両端のCHR$(&H7F)をはずせば、残りは２バイトになり、CVI()にかけて整数型配列に格納できます。この整数型配列は③でも使えます(取り出した要素をMKI$()にかけたあとで、前後にCHR$(&H7F)を付け加えれば日本語文字になる)ので、この整数型配列を使った方法がよいでしょう。

③、④に共通するサンプルプログラムとして、１バイト系を２バイト系に変換し、それをまた１バイト系に変換するプログラムを紹介します。サブルーチン＊MJ1.TO.MJ２で１バイト系を２バイト系に変換し、＊MJ2.TO.MJ1で２バイト系を１バイト系に変換します。

リスト　8-7

```
100 REM
110 REM   1バイト系  ←→  2バイト系  変換サンプル
120 REM
130 '
140 REM ---- 変換表の初期化
150 DEFINT A-Z
160 DIM AKCNV(255)
170 FOR I=0 TO 255
180   READ D
190   IF D AND &HFF THEN D=D OR &H8080
200   AKCNV(I)=D
210 NEXT
220 '
230 REM ---- メイン処理
240 FOR I=0 TO 10
250   MJ1$=CHR$(RND*216+32)
260   GOSUB *MJ1.TO.MJ2
270   PRINT MJ1$" → "MJ2$;
280   GOSUB *MJ2.TO.MJ1
290   PRINT " → "MJ1$
300 NEXT
310 END
320 '
330 REM ---- 変換サブルーチン
340 '
350 *MJ1.TO.MJ2
360  MJ2=AKCNV(ASC(MJ1$))
370  MJ2$=CHR$(&H7F)+MKI$(MJ2)+CHR$(&H7F)
380 RETURN
390 '
400 *MJ2.TO.MJ1
410  MJ2=CVI(MID$(MJ2$,2,2))
```

```
420  MJ1=SEARCH(AKCNV,MJ2)
430  IF MJ1<0 THEN MJ1$="" : RETURN
440  MJ1$=CHR$(MJ1)
450 RETURN
460 '
470 REM ---- 変換表データ
480 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
490 DATA &H2322, &H2322, &H0A00, &H2322, &H2322, &H0D00, &H2322, &H2322
500 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
510 DATA &H2322, &H2322, &H2322, &H1B00, &H2A22, &H2B22, &H2C22, &H2D22
520 DATA &H2121, &H2A21, &H4921, &H7421, &H7021, &H7321, &H7521, &H4721
530 DATA &H4A21, &H4B21, &H7621, &H5C21, &H2421, &H5D21, &H2521, &H3F21
540 DATA &H3023, &H3123, &H3223, &H3323, &H3423, &H3523, &H3623, &H3723
550 DATA &H3823, &H3923, &H2721, &H2821, &H6321, &H6121, &H6421, &H2921
560 DATA &H7721, &H4123, &H4223, &H4323, &H4423, &H4523, &H4623, &H4723
570 DATA &H4823, &H4923, &H4A23, &H4B23, &H4C23, &H4D23, &H4E23, &H4F23
580 DATA &H5023, &H5123, &H5223, &H5323, &H5423, &H5523, &H5623, &H5723
590 DATA &H5823, &H5923, &H5A23, &H4E21, &H6F21, &H4F21, &H2322, &H3221
600 DATA &H2322, &H6123, &H6223, &H6323, &H6423, &H6523, &H6623, &H6723
610 DATA &H6823, &H6923, &H6A23, &H6B23, &H6C23, &H6D23, &H6E23, &H6F23
620 DATA &H7023, &H7123, &H7223, &H7323, &H7423, &H7523, &H7623, &H7723
630 DATA &H7823, &H7923, &H7A23, &H5021, &H4321, &H5121, &H4121, &H2322
640 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
650 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
660 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
670 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
680 DATA &H2322, &H2321, &H5621, &H5721, &H2221, &H2621, &H7225, &H2125
690 DATA &H2325, &H2525, &H2725, &H2925, &H6325, &H6525, &H6725, &H4325
700 DATA &H3C21, &H2225, &H2425, &H2625, &H2825, &H2A25, &H2B25, &H2D25
710 DATA &H2F25, &H3125, &H3325, &H3525, &H3725, &H3925, &H3B25, &H3D25
720 DATA &H3F25, &H4125, &H4425, &H4625, &H4825, &H4A25, &H4B25, &H4C25
730 DATA &H4D25, &H4E25, &H4F25, &H5225, &H5525, &H5825, &H5B25, &H5E25
740 DATA &H5F25, &H6025, &H6125, &H6225, &H6425, &H6625, &H6825, &H6925
750 DATA &H6A25, &H6B25, &H6C25, &H6D25, &H6F25, &H7325, &H2B21, &H2C21
760 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
770 DATA &H2322, &H2322, &H2322, &H2322, &H7C21, &H7B21, &H2322, &H2322
780 DATA &H2322, &H5F31, &H2F47, &H6E37, &H7C46, &H7E3B, &H2C4A, &H4349
790 DATA &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322, &H2322
```

第9章　フロッピーディスク

9-1　はじめに

この文章はN88DISK - BASICのDISKに関する部分について述べてありますが、その前におことわりしておかなければならないことがあります。

①本書で述べるN88DISK-BASICは、システム起動時に次のように表示されるものです。

Disk version ［Aug 19, 1983］
How many files(0-15)?

これ以前の日付のバージョン(旧PC-8801用)ではアドレスが一部異なる場合があります。

②用語の問題

- 両面ディスクの場合、表裏の2つのトラックを合わせてシリンダと呼ぶことがありますが、本章(の本文)では使用していません。
- この章では、次のものを同義語として使っています。
  フロッピーディスクの同義語：ディスク、ディスケット
  ディスクドライブの同義語：ドライブ
- トラック番号が示すもの
  トラック番号が示すものには2種類あります。ひとつはBASICレベルで使われている考え方で、あるトラックを示すのに、サーフェス番号とトラック番号を使うもので、トラック番号だけでは表裏両方のトラックを指します。この場合、トラック番号はシリンダ番号と同じになります。
  もうひとつはサーフェス番号をトラック番号の中に取り込んだ考え方で、シリンダ0表、裏、シリンダ1表、裏、……と順にトラック番号をつけていきます。この方法だと、ひとつのトラック番号でひとつのトラックを指せるので、つごうがよく合理的で、BASICでも内部処理上はこちらを使用しています。しかし、片面で使用するときと、両面で使用するときとで、同じトラックを指すのにトラック番号が違ってきて、直感的にわかりにくくなります。

この章はBASICからみたディスクについて説明しますので、前者の方法を使います。
第10章では内部処理に立ち入りますので、後者を使います。

9-2　ディスクの構造

N88DISK-BASICで使用できる３種類のディスクの構造について説明します。これは正常のN88-DISK-BASICで使われている構造で、他のOSや、コピープロテクトされたディスクでは異なる場合があります。

9-2-1　ディスク・マップ

①8インチ両面

PC-8881を接続すると使えるようになります。

トラック数　片面77　(使用しているのは片面76)
記録方式　MFM
セクタ数　26セクタ／トラック
セクタ長　256バイト／セクタ
データ容量　1,011,712バイト(フォーマット後.システム領域を含む)

サーフェス0

トラック	セクタ	クラスタ	用　途
0 (0H)	1~26	0	未使用
1 (0H)	1	2	IPL
1 (1H)	2~26	2	DISKコード
2 (2H)	1~26	4	DISKコード
35(23H)	1~22	70(46H)	ディレクトリ
35(23H)	23	70(46H)	ID
35(23H)	24~26	70(46H)	FAT(0~99H)

ユーザ領域
トラック　3~34
トラック　36~76

サーフェス1

トラック	セクタ	用　途
0	1	未使用
1	3	DISKコード

ユーザ領域
トラック　2~76

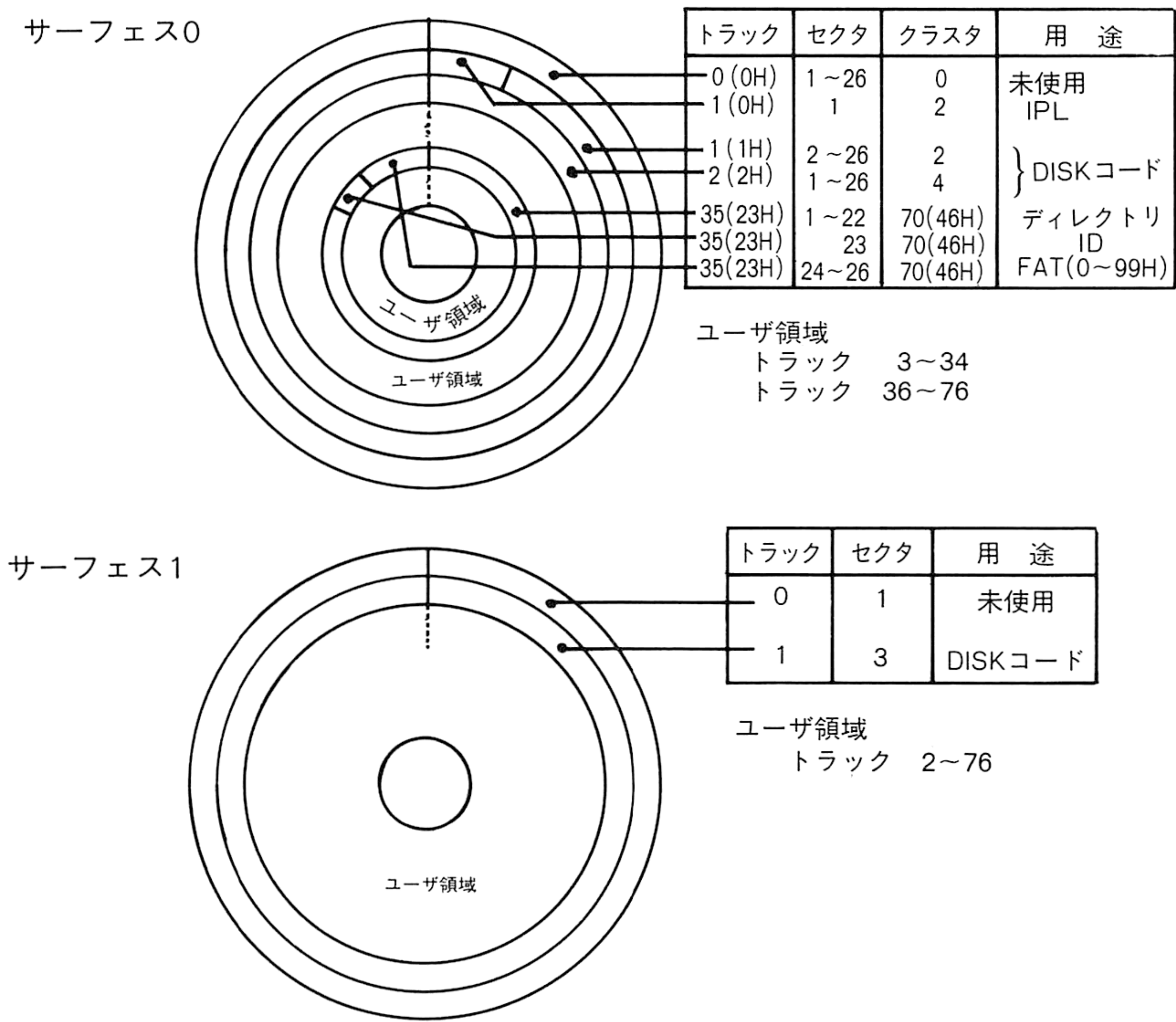



図9-2-A　8インチ両面ディスクマップ

②5インチ両面

内蔵のドライブで使うタイプで、本章でもこのタイプを前提にしています。

トラック数　片面40
記録方式　　MFM
セクタ数　　16セクタ／トラック
セクタ長　　256バイト／セクタ
データ容量　327,680バイト(フォーマット後.システム領域を含む)

サーフェス0

トラック	セクタ	クラスタ	用　途
0	1	0	IPL
0	2～16	0，1	DISKコード
1	1～16	4，5	DISKコード

ユーザ領域
トラック　2～39

ユーザ領域

サーフェス1

トラック	セクタ	クラスタ	用　途
0	1～16	2，3	DISKコード
1	1～16	6，7	DISKコード
18(12H)	1～12	74，75	ディレクトリ
18(12H)	13	75(4BH)	ID
18(12H)	14～16	75(4BH)	FAT(0～9FH)

ユーザ領域
トラック　2～17
トラック　19～39

ユーザ領域

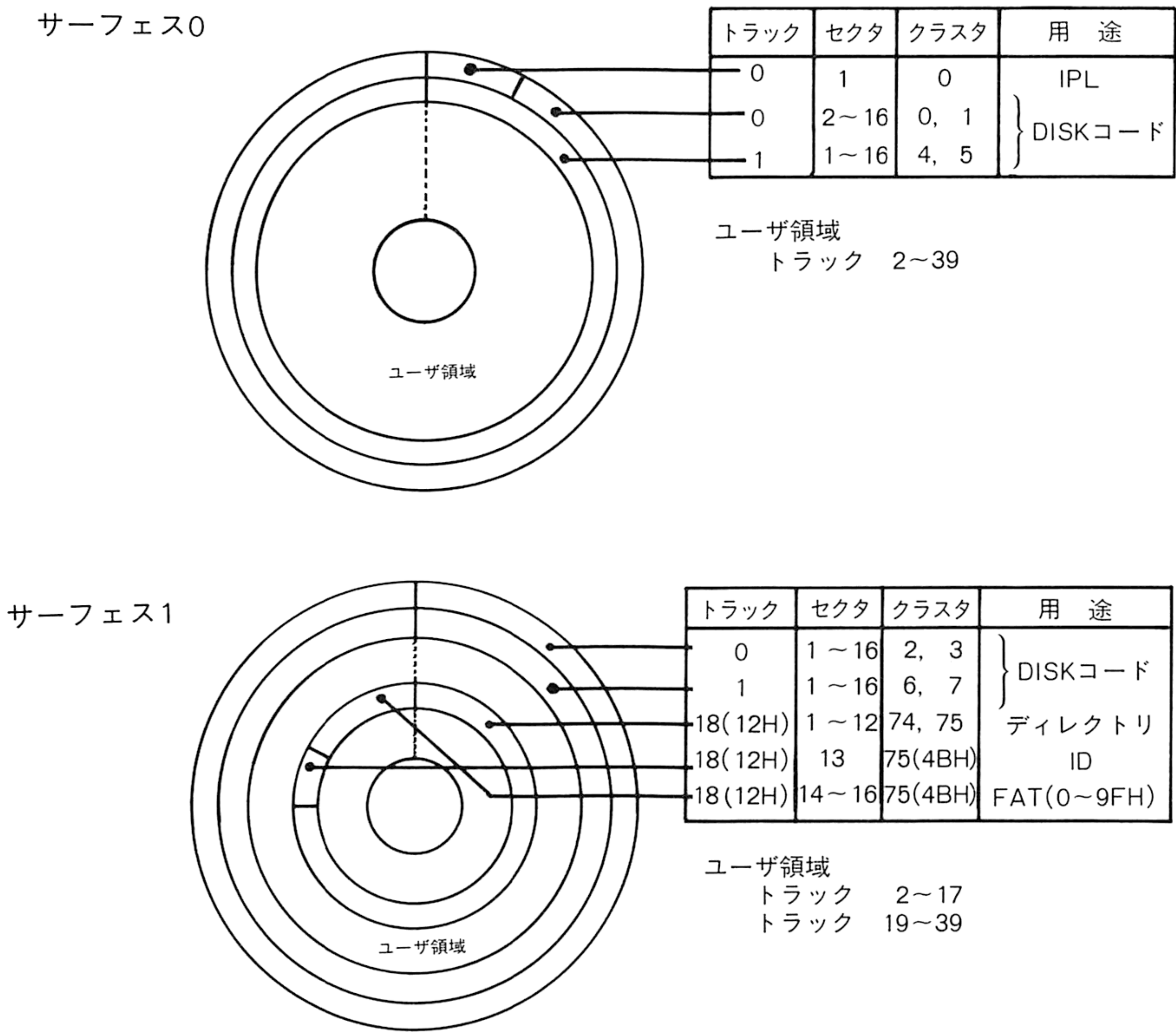



図9-2-B　5インチ両面ディスクマップ

③5インチ片面

PC-8031で使用するタイプですが、PC-8801mkⅡには接続できませんので、通常は使用しません。しかし、ソフトウエアによって、内蔵ドライブでもこのタイプを使用できるようになります。(第15章ランダムテクニック参照)

トラック数　35
記録方式　　MFM
セクタ数　　16セクタ／トラック
セクタ長　　256バイト／セクタ
データ容量　143,360バイト(フォーマット後.システム領域を含む)

トラック	セクタ	クラスタ	用　途
0	1	0	IPL
0	2～16	0，1	DISKコード
1	1～16	2，3	
2	1～16	4，5	
3	1～16	6，7	
18(12H)	1～12	36, 37	ディレクトリ
18(12H)	13	37(25H)	ID
18(12H)	14～16	37(25H)	FAT(0～45H)

ユーザ領域
ユーザ領域

ユーザ領域
トラック　4～17
トラック　19～34

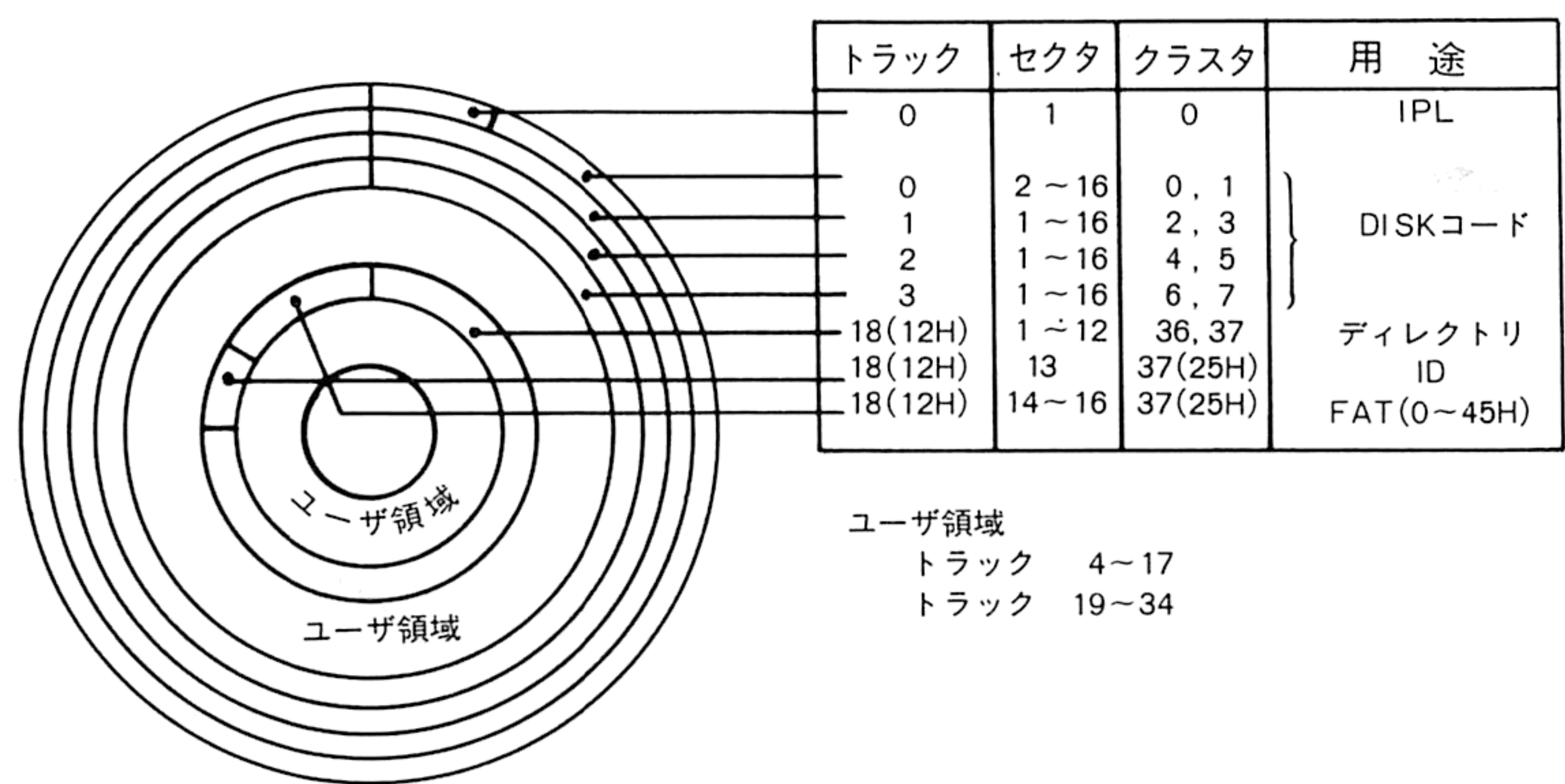



図9-2-C　5インチ片面ディスクマップ

9-2-2　　ディスクアドレスとクラスタとの変換

1)5インチ片面のとき　1クラスタ＝8セクタ

＜クラスタ＞＝＜トラック＞×2＋＜セクタ＞¥9

＜トラック＞＝＜クラスタ＞¥2

＜セクタ＞＝(＜クラスタ＞　MOD　2)×8＋1から8セクタ

2)5インチ両面のとき　1クラスタ＝8セクタ

＜クラスタ＞＝＜トラック＞×4＋＜サーフェス＞×2＋＜セクタ＞¥9

＜トラック＞＝＜クラスタ＞¥4

＜サーフェス＞＝(＜クラスタ＞　MOD　4)¥2

＜セクタ＞＝(＜クラスタ＞　MOD　2)×8＋1　から8セクタ

3)8インチ両面のとき　1クラスタ＝26セクタ

＜クラスタ＞＝＜トラック＞×2＋＜サーフェス＞

＜トラック＞＝＜クラスタ＞¥2

＜サーフェス＞＝＜クラスタ＞MOD　2

＜セクタ＞＝1セクタ～26セクタまで全部

9-2-3　　ディレクトリ

ディレクトリには、ファイル名、ファイル属性、ファイルが格納されている先頭クラスタ番号がしまわれています。これによりファイル名とファイルが格納されている場所との対応がつけられます。

ディレクトリの位置は、

5インチ片面……トラック18，セクタ1～12

5インチ両面……サーフェス1，トラック18，セクタ1～12

8インチ両面……サーフェス0，トラック35，セクタ1～22

となっています。1つのファイルに対して16バイトが割り当てられていますが、そのうち使用されているのは12バイトです。ディレクトリはこのようになっています。

5インチ両面のとき

```
h〕^d1,1,12,1
Track 12,Surface 01,Sector 01
0000 63 72 74 73 65 74 6E 38 38 80 48 FF FF FF FF FF      crtsetn88_H
0010 00 62 70 72 6E 74 6E 38 38 80 47 FF FF FF FF FF       bprntn88_G
0020 45 46 49 4C 45 53 6E 38 38 C0 49 FF FF FF FF FF      EFILESn88ﾀI
0030 46 45 44 49 54 20 6E 38 38 80 46 FF FF FF FF FF      FEDIT n88_F
0040 00 61 69 6A 69 49 6E 38 38 80 45 FF FF FF FF FF       aijiIn88_E
0050 48 45 58 44 54 41 6E 38 38 90 4C FF FF FF FF FF      HEXDTAn88┴L
0060 73 70 6F 6B 65 20 6E 38 38 80 45 FF FF FF FF FF      spoke n88_E
0070 4D 41 43 48 4E 4C 62 69 6E 01 42 FF FF FF FF FF      MACHNLbin.B
0080 54 43 48 4E 4F 57 64 74 61 00 47 FF FF FF FF FF      TCHNOWdta G
0090 50 52 54 45 43 54 6E 38 38 A0 41 FF FF FF FF FF      PRTECTn88 A
00A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
h〕
```

拡張子

未使用

ファイル名

先頭バイト

00：killされたファイル

FF：未使用

ファイルが格納されている

先頭クラスタ番号

属　性

ビット　7　6　5　4　3　2　1　0

機械語ファイル

ライトプロテクト（P）

Pオプションファイル（E）

リードアフターライト（R）

非アスキー形式

図9-2-D　ディレクトリ

9-2-4　IDセクタ

IDにはユーザーズマニュアルにあるとおり、ディスク全体の属性、一度にOPENできるファイルの数、電源ON(リセット)時に実行される文が格納されています。１度にOPENできるファイル数と、電源ON時に実行される文は、システムディスクでないと意味を持ちません。

IDは、

5インチ片面……トラック18，セクタ13

5インチ両面……サーフェス1，トラック18，セクタ13

8インチ両面……サーフェス0，トラック35，セクタ23

に割り当てられています。

IDセクタの構成はこうなっています。

属　性	ファイル数	BASIC テキスト
1バイト	1バイト	254バイト

図9-2-E　IDセクタの構成

ところでディスクユーティリティーの「SET UP INFO」には年を設定する機能がありましたが、これもこのセクタに書き込まれます。ではどこに格納されるのでしょうか。これはBASICテキストの中に、POKE文として書き込まれるのです。たとえば、年を85年として設定したディスクのIDセクタを見てみると、

```
h)^d1,1,12,d（5インチ両面の場合）
Track 12,Surface 01,Sector 0D
0000 00 03 70 6F 6B 65 20 26 48 66 30 31 32 2C 26 48   .poke &Hf012,&H
0010 38 35 3A 70 72 69 6E 74 22 66 69 6C 65 73 22 3A   85:print"files":
0020 66 69 6C 65 73 00 00 00 00 00 00 00 00 00 00 00   files
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
h)
        1度にOPENできるファイルの数
     属性（ディレクトリと同じ）
```

図9-2-F　IDセクタの内容

となります。POKE &HF012, &H85というのがそれです。F012H番地にはDATE$の年の情報が格納されています。ここに&H85を入れると年が85年になるわけです。

9-2-5　FAT(File Allocation Table)

FATはファイルの格納状態を示します。ファイルが1クラスタに納まらない場合、残りを別のクラスタに書き込まなければなりません。この「別のクラスタ」をどこにするかには、いろいろな方法がありますが、N88DISK-BASICでは、適当に空いているクラスタに書き込みます。(規則性はありますが、本質的にはどの空きクラスタに書き込もうとかまいません。) このとき、どこのクラスタに書き込んだかを記録しておかないと、後で困ることになります。また、「別のクラスタ」を捜す時に、どのクラスタが空いているかが分からなければなりません。これらの情報を記録したものが、FATです。

ではこのFATはどのようになっているか見てみましょう。

FATの位置は、

5インチ片面……トラック18, セクタ14〜16, 160バイト

5インチ両面……サーフェス1, トラック18, セクタ14〜16, 70バイト

8インチ両面……サーフェス0, トラック35, セクタ24〜26, 154バイト

となっていて、3つのセクタとも同じものが入っています。

```
h)^d1,1,12,e（5インチ両面の場合）
Track 12,Surface 01,Sector 0E
0000 FE FE FE FE FE FE FE FE FF FF FF FF FF FF FF FF ┐
0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF │
0020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF │
0030 FF FF FF FF FF FF FF FF C2 FF FF 38 3B FF 3C C6 │
0040 3F C1 40 3E C8 C8 44 43 C5 C8 FE FE 4D C1 FF FF │ FAT
0050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF │ 160バイト
0060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF │
0070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF │
0080 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF │
0090 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ┘
00A0 C6 87 E5 00 45 45 00 01 00 00 00 00 00 00 00 00 ┐
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 │
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 │ 意味は
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 │ ありません
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 │
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ┘
h)   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
```

図9-2-G　FATの内容

クラスタ47Hを見て下さい。それはTCHNOWdtaというファイルの先頭クラスタです。ここには43Hという値が入っています。これはデータがクラスタ47Hに入り切れず、クラスタ43Hに続いていることを示します。クラスタ43Hを見ると3EHとなっています。こうして

47H→43H→3EH→3CH→3BH→38H

と続いていきます。クラスタ38Hを見ると、値はC2Hとなっています。クラスタC2Hというものはありません。値がC1H〜C8H(8インチの時はC1H〜DAH)の時は、ここのクラスタでファイルが終わっていて、下位5ビット(5インチの時1〜8, 8インチの時1〜1AH)がそのクラスタで実際に使用しているセクタ数を表わします。ここではクラスタ38Hのうち、2セクタを使用して、ファイルが終わっていることを示します。

これらをまとめると次のようになります。

バイトのデータ（16進）	クラスタの使用状態
8インチ両面の時　0〜99 5インチ両面の時　0〜9F 5インチ片面の時　0〜45	使用中。連続したクラスタの一部であり、後続するクラスタを持つ。値が、後続するクラスタの番号を示している。
8インチ両面の時　C1〜DA 5インチの時　C1〜C8	使用中。連続したクラスタの最後のクラスタであり、下5（5インチの時は下4）ビットの内容が、そのクラスタで実際につかわれているセクタの数を表わす。
FE	予約済みのクラスタで、ファイルとして使うことはできない。(DISKコード、IPL、ディレクトリ、FAT自身を含むクラスタがこれである。)
FF	未使用。

9-3　ドライブテーブル

ドライブポインタとドライブテーブルは、電源ON(リセット)時に、接続されているドライブの数だけ確保されます。各ドライブポインタ(2バイト)は、対応するドライブテーブルの、先頭から3バイト目のアドレスを持っています。

各ドライブテーブルの構造は図9-3-Bのとおりです。ドライブテーブルの先頭バイトがFFHの時は、ディスク上のFATがまだ更新されていませんので、そのドライブのディスクを抜いてはいけません。必ず、CLOSEまたはENDを行なってから抜いて下さい。また、ディスクにファイルをOPENした時も、必ずCLOSEまたはENDを行なってからでないと、ディスクを抜いてはいけません。これはN-DISK-BASICのREMOVEに相当するのです。

8000
テキストウィンドウ
8400
ディスクコード
AE36
ドライブポインタ#1　←(EC81,82)
ドライブポインタ#2
⋮
ドライブポインタ#n
ファイルポインタ
ドライブテーブル#1
ドライブテーブル#2
⋮
ドライブテーブル#n
ファイルバッファ
n ：接続されているドライブ数
EC7DH番地に入っている

図9-3-A　ドライブテーブルの位置

オフセット	内容
−2	ディスク上のFATの更新が 必要な時………FF 必要でない時…00
−1	IDにある属性
ドライブポインタが指すアドレス→	ディスク上にファイルがOPENされている時、最後にアクセスしたトラック番号
+1	
+2	未使用のクラスタ数
+3 ～ +163	ドライブに入っているディスクのFAT 使用しているのは 5インチ片面のとき 70バイト 5インチ両面のとき160バイト 8インチのとき 154バイト

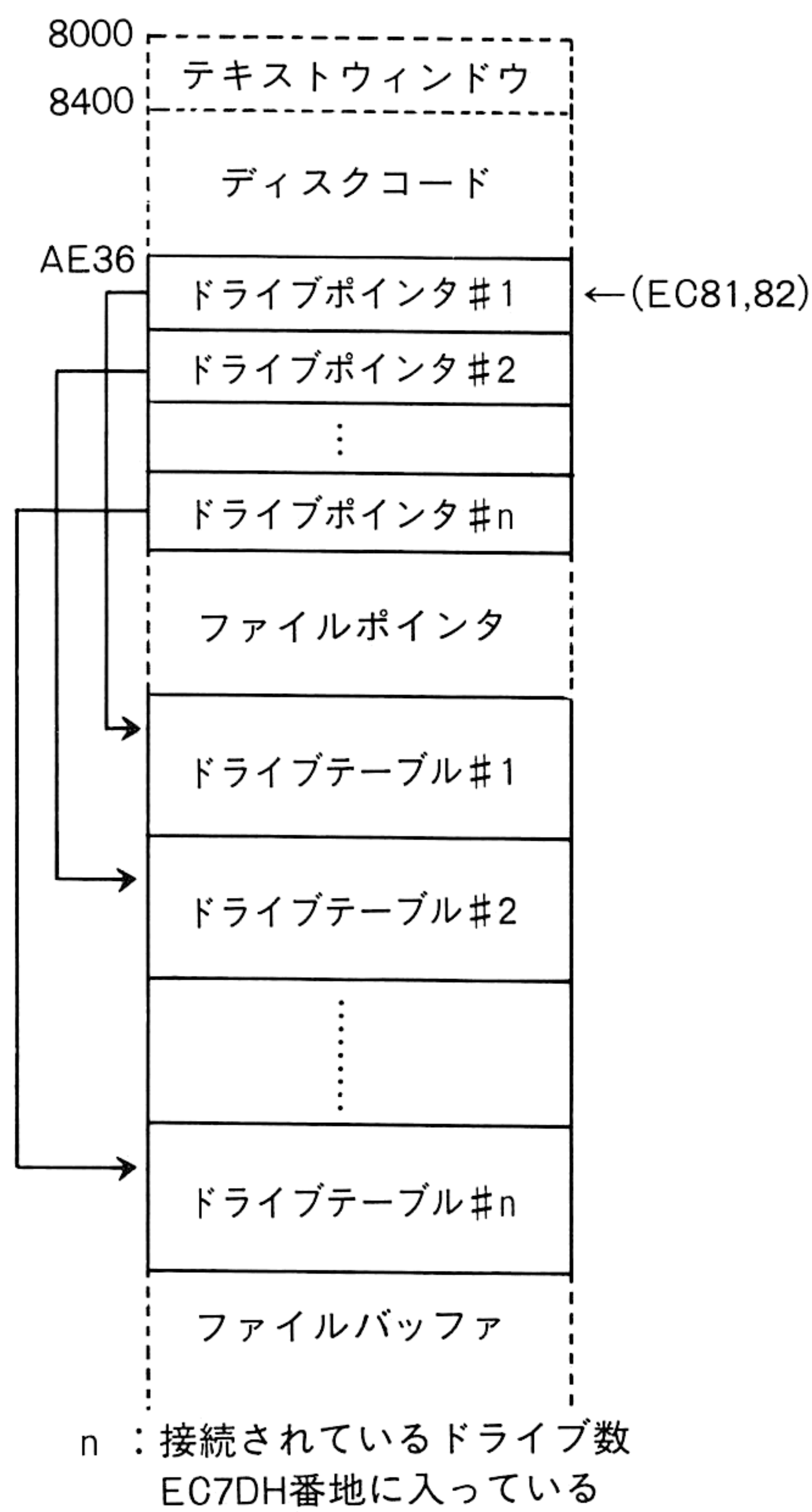
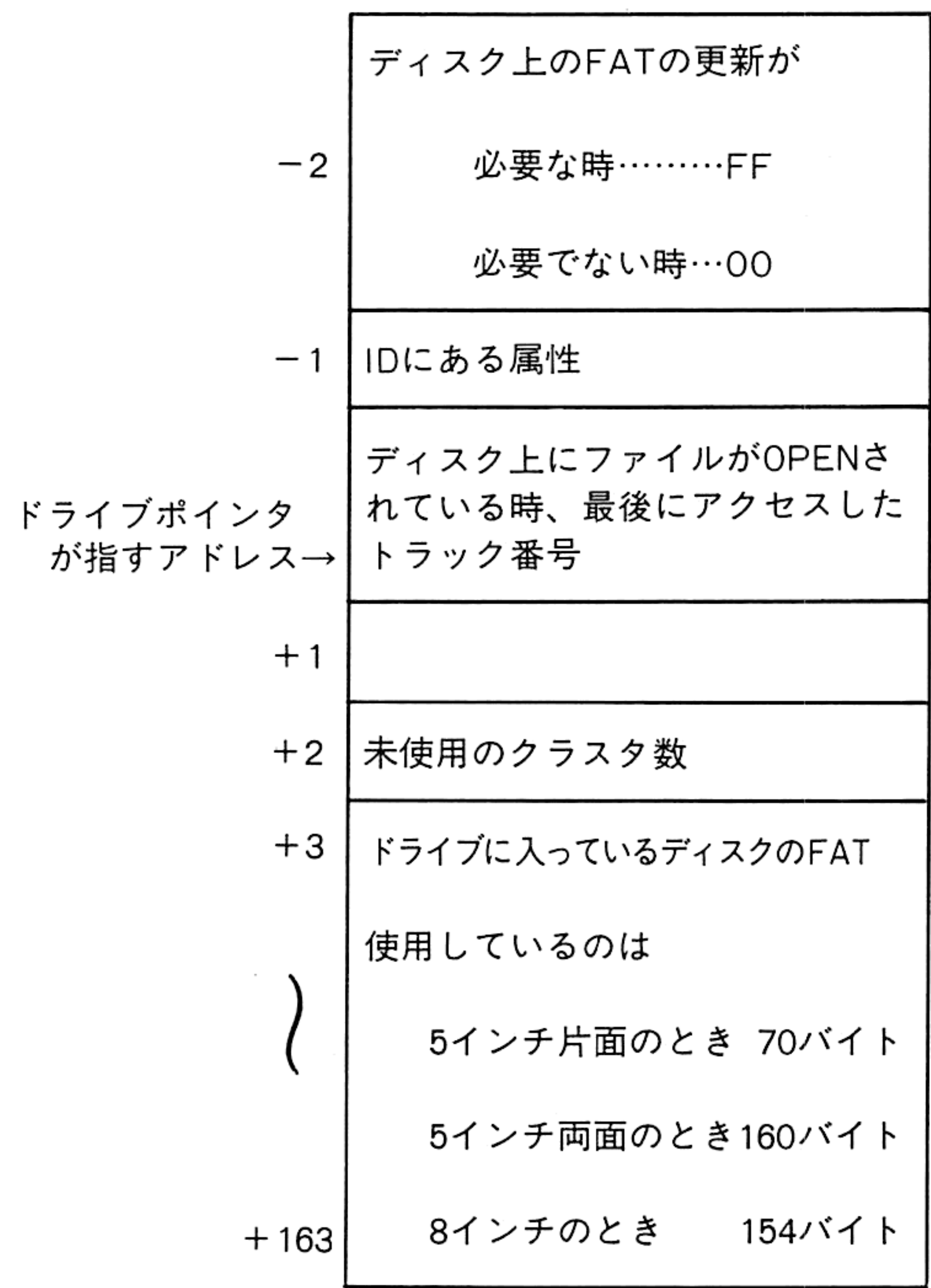



図9-3-B　ドライブテーブルの構造

9-4　DSKF関数

DSKF関数は、ディスクファイルの構造に関する情報を返す関数で、機能を指定することによって、別表のような値が得られます。

この値のデータはN88DISK-BASIC上の8A18H番地に格納されています。

リスト　9-1

```
8A18 4C 1A 01 01 9A 23 1A 18 1A 03 17 34
8A24 22 10 00 02 46 12 08 0E 10 03 0D 10
8A30 27 10 01 02 A0 12 08 0E 10 03 0D 10
```

マニュアルでは11種類の機能しかあげてありませんが、上のデータを見ると、各ドライブタイプについて、12個ずつの値があるようです(DSKF機能一覧表)。

それからドライブタイプとしては、表の3つの他にDMA方式のミニフロッピーディスクドライブも考えられているようですが、これについては内蔵ミニフロッピードライブと同じようになっています。

DSKF機能一覧表

機能番号	8—2D		5—2D		5—1D		機　　　能
	16進	10進	16進	10進	16進	10進	
0	4C	76	27	39	22	34	片面あたりの最大トラック番号
1	1A	26	10	16	10	16	1トラックあたりのセクタ数
2	01	1	01	1	00	0	片面…0、両面…1
3	9A	1	02	2	02	2	1トラックあたりのクラスタ数
4	9A	154	A0	160	46	70	クラスタの総数
5	23	35	12	18	12	18	ディレクトリが格納されているトラック番号
6	1A	26	08	8	08	8	1クラスタあたりのセクタ数
7	18	24	0E	14	0E	14	FATの開始セクタ番号
8	1A	26	10	16	10	16	FATの終了セクタ番号
9	03	3	03	3	03	3	FATの個数
10	17	23	0D	13	0D	13	IDが格納されているセクタ番号
11*	34	52	10	16	10	16	1クラスタあたりのセクタ数×2

(＊N88-DISK-BASICでは使われていません。)

9-5　標準ディスク

9-5-1　インタリーブ13フォーマッティング

市販されている標準(8インチ)のフロッピーディスクを使用すると、NECのディスクよりも入出力が遅いことがあります。これはディスクの物理的フォーマットの方法が異なるためです。

NECのディスク(PC-8886)と他社の多くのディスクのフォーマットの違いは次のようなものです。一般的なディスクは、トラック上にセクタが1から26まで順番に並んでいます。ところが、NECのディスクの場合には、セクタが1つおきに続いています。この形式をインタリーブ13と呼びます。第1セクタの次が第14セクタで、差が13だからです。なぜこのようなフォーマットにしたのかというと、PC-8881を使用して、連続したセクタを読み書きする時には、この方が速いからです。

あるトラックの第1セクタ～第3セクタを続けて読む場合を考えます。第1セクタを読み取った後、第2セクタを読みに行くまでの間には、いろいろな処理が必要です。このため、セクタが順に並んでいる場合には、第2セクタを読み取ろうとした時にはすでに第2セクタの先頭がヘッドを行きすぎてしまい、第2セクタがぐるっと1周してくるまで待たなくてはなりません。NECディスクの形式では、第1セクタの次は第14セクタですから、第14セクタが行き過ぎるまで待てばよいわけで、このほうがずっと速いわけです。

もちろん1セクタだけのアクセスであれば、どちらの形式でも違いはありません。なお、このフォーマットの違いは、あくまでも物理的なもので、通常な使用法ではどちらもまったく同じに扱えます。

インタリーブ13形式でフォーマットするには、システムディスクのディスクユーティリティーで物理的フォーマットを指定します。

9-5-2　トラック0

標準ディスクの場合、N88DISK-BASICではトラック0が表裏とも使用されていません。これはなぜかというと、トラック0は他のトラックとは物理的なフォーマットが違っているからです。モニタで読んでみようとしても、まったく読めません。

トラック0のサーフェス0は単密度(FM方式、128バイト／セクタ)でフォーマットされています。そのため通常の方法では読み書きできません。ぜひとも読みたい場合は、μPD765を直接コントロールするプログラムを書く必要があります。

トラック0の内容は通常次のようになっています。

セクタ5：エラーマップ

フォーマット時に不良トラックの情報が書き込まれています。

セクタ7：ボリュームラベル

ボリューム名やユーザーIDのほか、他のトラックのセクタ長やセクタの並んでいる順序などの情報が入っています。

セクタ8～26：ファイルラベル

ファイル名やその場所が書かれます。IBM方式のファイルのディレクトリです。

トラック０，　サーフェス０

セクタ	
セクタ０１	IPL用
０２	IPL用
０３	スクラッチ用
０４	予　備
０５	エラーマップ
０６	予　備
０７	ボリュームラベル
０８	ファイルラベル１
０９	ファイルラベル２
≈	≈
２６	ファイルラベル19

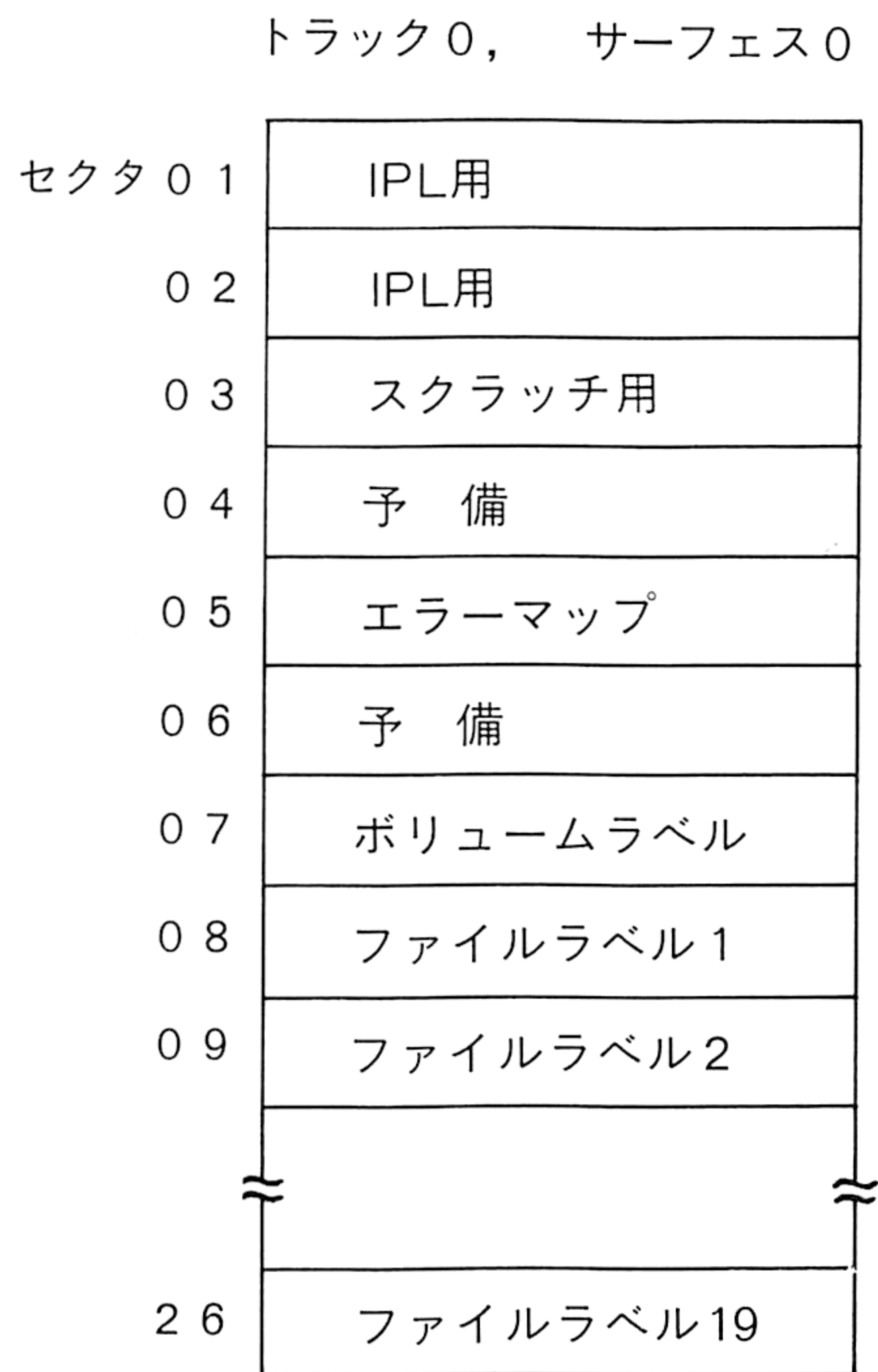



図9-5-A　トラック０、サーフェス０

これらの情報は、ディスクをIBM方式で使用する時に必要なもので、EBCDICコード(IBMの文字コード)を使用していて、N$_{88}$DISK-BASICではまったく使用されていません。それだけでなく、これらの情報も書き込まれてないディスクもあります。

トラック０でもサーフェス１は倍密度で書かれているのですが、読めない場合があります。ですが強引に読むことはできます。モニタのヘrコマンドを使うと、エラーにはなりますが、メモリ上にはちゃんと読み込まれています。これは、そのセクタにDDAM(Deleted Data Address Mark)という印がついているからです。また、書き込みは自由です。いったん書き込むとその後は正常に読むことができます。

9-6　BASICによるユーティリティ

ここではBASICで作った5つのディスクユーティリティを紹介します。プログラムリストは章末にまとめて掲載してあります。

9-6-1　拡張FILES

ファイル名とファイルの大きさだけでなく、その属性や、使用しているクラスタ番号、また機械語ファイルの場合には、そのアドレスも出力する「拡張FILES」をBASICで作った例を紹介します。実行例を下に示します。

リスト　9-2

```
CRT (c) or PRINTER (p) ? c
DRIVE NUMBER ? 1

         DRIVE 1             FREE 133

FILE-NAME   ATTR  ADDRESS    LOCATION
crtset.n88   N    ----------  48(5)
EFILES.n88   R    ----------  49(8)
FEDIT .n88   N    ----------  46 44(8)
HEXDTA.n88   P    ----------  4C 4D(1)
spoke .n88   N    ----------  45(8)
MACHNL*bin   N    D000-E561   42 40 3F(6)
TCHNOW dta   N    ----------  47 43 3E 3C 3B 38(2)
PRTECT.n88   E    ----------  41(1)
Ok
```

プログラムでは、ディレクトリをDSKI$関数で読んでいくことにより、ファイル名、属性、先頭クラスタ番号を得ています。ファイルが使用しているクラスタ番号は、FATを読んで、順に追ってゆけばわかります。最後のカッコの中の数値は、最後のクラスタの中で実際に使用しているセクタ数を表わします。

機械語ファイルの場合はアドレスを読んできます。アドレスは、ファイル自身の最初の4バイトに開始番地、終了番地＋1が書き込まれています。

9-6-2　ディスクエディット

ディスケットをセクタ単位で書き換える場合には、普通はモニタで次のようにします。

①へrでメモリ上に読み込む

②eでそのデータを修正する

③へwでディスクに書く

④へdコマンドで確かめる

しかし、へwで書く時に間違ったセクタに書き込んでしまったら大変です。またeコマンドでは16進または8進での修正しかできません。ここで紹介するプログラムは、読み込んだセクタに書き込むことを原則とし、文字での修正も可能なディスクエディタです。

RUNするとまず"Drive？" とたずねてきます。修正したいディスクのドライブ番号を入力して下さい。ここで０と入力すると、ディスクからでなく、メモリーからデータを読むことになります。

次にサーフェス、トラック、セクタを順次入力して下さい。

Drive？　＜ドライブ番号＞

１のとき	０のとき
＜サーフェス入力＞	＜アドレス入力＞
＜トラック入力＞	
＜セクタ入力＞	

下のように表示されます。

リスト　9-3

```
■■ DISK  EDITOR ■■   Drive 1  Surface 0  Track 24  Sector 3
    +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F   0123456789ABCDEF
0:  26 02 40 01 20 CB 20 15 20 3A 20 D6 20 47 58 31   &.@. .  : ∃ GX1
1:  20 20 2C 47 59 31 E3 13 20 3B 20 91 20 22 9B 95     ,GY1.. ; .  "..
2:  95 95 9B 22 00 36 02 4A 01 20 8D 20 F5 47 45 54   ..." [illegible] GET
3:  2E 53 55 42 00 46 02 54 01 20 82 20 49 E1 12 20   .SUB [illegible]
4:  DC 20 0E 14 00 63 02 5E 01 20 20 20 47 58 31 E1   [illegible] GX1.
5:  FF 00 E5 0F 46 20 3B 20 47 59 31 E1 EF 98 E5 0F   [illegible] GY1 [illegible]
6:  14 00 80 02 68 01 20 20 20 47 58 32 E1 47 58 31   [illegible] GX2.GX1
7:  E3 15 20 3B 20 47 59 32 E1 47 59 31 E3 13 00 92   . ; GY2.GY1 [illegible]
8:  02 72 01 20 20 20 8D 20 F5 50 55 54 2E 53 55 42   [illegible] PUT.SUB
9:  00 99 02 2C 01 20 83 00 9F 02 86 01 81 00 C3 02   [illegible]
A:  90 01 3B 8E E9 2D 2D 2D 2D 2D 2D 47 45 54 40 2D   [illegible] ------GET@-
B:  53 55 42 20 52 4F 55 54 49 4E 45 20 2D 2D 2D 2D   SUB ROUTINE ----
C:  2D 00 E7 02 9A 01 3B 8F F9 47 45 54 40 41 28 47   - [illegible] GET@A(G
D:  58 31 2C 47 59 31 29 2D 28 47 58 32 2C 47 59 32   X1,GY1)-(GX2,GY2
E:  29 2C 41 52 59 00 0B 03 A4 01 3A 8F E9 2D 2D 2D   ),ARY [illegible] ---
F:  2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D   ----------------
EDIT.SAVE.FIN(e/s/f) ?

 load"   files   go to   list   run
```

ここで次の３つを選びます。

EDIT	(e⏎と入力)．．．．	データの修正
SAVE	(s⏎　〃　)．．．．	データの書き込み
FIN	(f⏎　〃　)．．．．	終わり

EDITモードでは３つのファンクションキーに意味があります。

f・１：16進で入力する時に押します。最初はこのモードです。

f・２：文字で入力する時に押します。カーソルが右の文字が表示されている領域に移動します。

f・５：EDITモードから抜け出します。

9-6-3　ファイルソート

Filesで出てくるファイル名は、ディレクトリに登録されている順番で表示されます。これをABC順に並べ変えるプログラムを紹介しましょう。

RUNすると、ディレクトリを読み込んだ後、ソートします。ここで" MAY I WRITE

ON DISK？” ときいてきますので、yまたはnを入力して下さい。yを入力すると、ソートしたディレクトリをディスクに書き込みます。

9-6-4　ファイル・リロケーション

今度はファイルをABC順ではなく、自分の好きな順番に並べ変えようというものです。このプログラムで処理できるのは、ディレクトリの最初の5セクタです。最大80個のファイルの並べ変えができます。

RUNするとディレクトリを読み込んだ後、次のように表示されます。

リスト　9-4

```
DIRECTORY RELOCATION
 SECTOR   1       SECTOR   2       SECTOR   3       SECTOR   4       SECTOR   5
--------------------------------------------------------------------------------
 dskut2 n88       filsrt n88       ( Killed )       textGP n88       movram n88
 @load  n88       fkicmd n88       ram+75 n         epepo1 bin       epepo2 bin
 @exst            fkisub n88       agsmpl n88       recov  bin       lowres n88
 MEND             gaijiL n88       setshg n88       atrget bin       gvramd n88
 ack232 n88       gaijiM n88       rdtxtr n88       spoke  bin       colors n88
 ( Killed )       gaijil n88       pndemo n88       grphsb bin       kjput  n88
 ( Killed )       grdtgn n88       scrlpt n88       spgdws bin       memdp1 n88
 ( Killed )       ( Killed )       mjtojs n88       NB1200 n         memdp2 n88
 ( Killed )       ( Killed )       jstomj n88       RAM+75 n         memdp3 n88
 grphld n88       kjcfrm n88       akcnv  k88       HXBDIB n88       ( Empty )
 dskedt n88       kjchft n88       hsroll n88       SELXFR n88       ( Empty )
 efiles n88       kjjsrm n88       holdwd n88       txtcpy n88       ( Empty )
 errcrs n88       dmaon  n88       gpsmpl n88       grcpbg n88       ( Empty )
 errmes n88       knjchr n88       hscls  bin       grphs0 n88       ( Empty )
 fbprnt n88       kwlst1 n88       crtset n88       GVRAM  scr       ( Empty )
 filrlc n88       kwlst2 n88       symchr n88       undlin n88       ( Empty )
--------------------------------------------------------------------------------

      MARK         SWAP         RENAME                      END
```

(Killed)はkillされたファイルが入っていた所で、(Empty)はまだファイルが入っていないところです。右の、文字が反転している場所はカーソルです。

カーソルキーとファンクションキーを使ってファイルを移動します。カーソルキーを押すと、その方向にカーソルが移動します。

MARK　(f・1キー)を押すとその場所がブリンクします。

SWAP　(f・2キー)を押すと、カーソルのあるファイルとマークされたファイル(ブリンクしている)とが入れ換わります。

このMARKとSWAPでファイルの移動を行なっていくわけです。

RENAME　(f・3キー)は、カーソルのあるファイルの名前を変更するためのものです。

END(f・5キー)を押すと、次のように尋ねてきます。

” WRITE ON DISK？(y／n／r)：”

yを入力……移動したデータをディスクに書き込みます。

nを入力……ディスクに書き込まずに処理を終わります。(ファイルの位置は変わらなかったことになります。)

rを入力……エディットモードに戻ります。

9-6-5　選択的ファイル転送

ディスク上のファイルを別のディスクに転送したいことがよくあります。しかし、ディスクユーティリティーの「TRANSFER FILES」を使うと、すべてのファイルを転送してしまいます。というわけで、指定したファイルだけを転送するプログラムを作ってみました。

このプログラムでは、ディスクの最初の90個のファイル(通常は十分な数です)のなかから、カーソルキーで転送したいプログラムを指定します。

実行させると、まずSOURCE DRIVEとDESTINATION DRIVEを尋いてきますので、転送元と転送先のディスクをセットしてから、それぞれ転送元ドライブ番号と転送先ドライブ番号を入力してください。

すると転送元のディスクからディレクトリを読み込んでこのように表示します。

リスト　9-5

```
██ Selective File Transfer ██     Drive1 -> Drive2(FREE 89)
--------------------------------------------------------------------------
[illegible]   @load .n88     @exst *        MENU   .       ack232.n88
grphld.n88    dskedt.n88     efiles.n88     errcrs.n88     errmes.n88
fbprnt.n88    filrlc.n88     filsrt.n88     fkicmd.n88     fkisub.n88
gaijiL.n88    gaijiM.n88     gaijiI.n88     grdtgn.n88     kjcfrm.n88
kjchft.n88    kjjsrm.n88     dmaon .n88     knjchr.n88     kwlst1.n88
kwlst2.n88    ram+75 n       agsmpl.n88     setsng.n88     rdtxtr.n88
pndemo.n88    scrlpt.n88     mjtojs.n88     jstomj.n88     akcnv .k88
hsroll.n88    holdwd.n88     gpsmpl.n88     hscls *bin     crtset.n88
symchr.n88    textGP.n88     epepo1*bin     recov *bin     atrget*bin
spoke *bin    grphsb*bin     spgdws*bin     NB1200.n       RAM+75.n
HX8DTA.n88    SELXFR.n88     txtcpy.n88     grcopy.n88     grphsv.n88
GVRAM  scr    undlin.n88     movram.n88     epepo2*bin     lowres.n88
gvramd.n88    colors.n88     kjput .n88     memdp1.n88     memdp2.n88
memdp3.n88
--------------------------------------------------------------------------
      Cursor keys to move, Space to select,Return key to end

>> Select the files to transfer.



  load      auto      go to      list      run
```

画面左上の反転している部分がカーソルで、カーソルキーによって移動します。転送したいファイルのところに移動させてスペースキーを押すと、ファイル名が登録されて、[　]で囲まれます。登録されたファイルの位置でもう一度スペースキーを押すと、登録解除になります。

転送したいファイルをすべて指定したら、⏎キーを押してください。ファイルが順に転送されます。転送先に同じ名前のファイルがあると、転送されたファイルで置き換わります。また、画面右上にFREEと表示されているのは、転送先の残り容量です。ファイルを登録するたびに減っていき、あとどれくらいファイルを転送できるかがわかります。ただし、これは正確な残り容量ではなく、転送先に同じ名前のファイルがあるときは、実際の残り容量よりも小さな値を示します。しかしFREEが0以上あれば、DISK FULLになることはありませんから、この値を目安に転送するファイルを指定することができます。

9-6-1　拡張ファイルズ　リスト

```
100 '
110 '   EXPANDED  FILES
120 '
130 DEFINT A-Z
140 PRINT : PRINT "CRT (c) or PRINTER (p) ? "; : E$=INPUT$(1) : PRINT E$
150 IF E$<>"c" AND E$<>"p" THEN 140
160 IF E$="c" THEN DEVICE$="scrn:" ELSE DEVICE$="lpt1:"
170 PRINT "DRIVE NUMBER ? "; : E$=INPUT$(1) : PRINT E$
180 DRIVE=VAL(E$) : IF DRIVE<1 OR DRIVE>PEEK(&HEC7D) THEN 170
190 OPEN DEVICE$ FOR OUTPUT AS#1
200 '
210 DIM FILE$(15),EX$(15),ATR$(15),CL$(15),FAT(DSKF(DRIVE,4)-1)
220 FOR I=0 TO 15
230   FIELD#0,I*16 AS DM$,6 AS FILE$(I),3 AS EX$(I),1 AS ATR$(I),1 AS CL$(I)
240 NEXT
250 DTYPE=3+(DSKF(DRIVE,1)=26)-DSKF(DRIVE,2)
260 IF DTYPE=1 THEN DM$=DSKI$(DRIVE,0,35,24) '8 inch
270 IF DTYPE=2 THEN DM$=DSKI$(DRIVE,1,18,14) '5 inch DS
280 IF DTYPE=3 THEN DM$=DSKI$(DRIVE,18,14)   '5 inch SS
290 FOR I=0 TO DSKF(DRIVE,4)-1
300   FAT(I)=ASC(MID$(DM$,I+1,1))
310   IF FAT(I)=255 THEN FREE=FREE+1
320 NEXT
330 '
340 PRINT
350 PRINT #1,"■■■■■■■ DRIVE";DRIVE;" ■■■■■■■■";
360 PRINT #1,"    FREE";FREE
370 PRINT #1,
380 PRINT #1,"FILE-NAME  ATTR  ADDRESS   LOCATION"
390 FOR SECTOR=1 TO DSKF(DRIVE,10)-1
400   IF DTYPE=1 THEN DM$=DSKI$(DRIVE,0,35,SECTOR)
410   IF DTYPE=2 THEN DM$=DSKI$(DRIVE,1,18,SECTOR)
420   IF DTYPE=3 THEN DM$=DSKI$(DRIVE,18,SECTOR)
430   FOR I=0 TO 15
440     P=ASC(FILE$(I))
450     IF P=0   THEN *NEXT.FILE
460     IF P=255 THEN END
470     ATR=ASC(ATR$(I)) : ATR$=""
480     IF ATR AND &H80 THEN FTYPE$="." ELSE FTYPE$=" "
490     IF ATR AND 1    THEN FTYPE$="*"
500     IF ATR AND &H40 THEN ATR$="R"
510     IF ATR AND &H20 THEN ATR$=ATR$+"E"
520     IF ATR AND &H10 THEN ATR$="P"
530     IF ATR$="" THEN ATR$="N"
540     ADRS$=STRING$(9,"-")
550     IF FTYPE$="*" THEN GOSUB *GET.ADRS
560     PRINT #1,FILE$(I);FTYPE$;EX$(I);"  ";ATR$;"   ";ADRS$;"  ";
570     CL=ASC(CL$(I))
580     WHILE CL<&HC0
590       PRINT #1," ";RIGHT$("0"+HEX$(CL),2);
600       CL=FAT(CL)
610     WEND
620     PRINT #1,"(";MID$(STR$(CL-&HC0),2);")"
630   *NEXT.FILE
640   NEXT
650 NEXT : END
660 '
670 *GET.ADRS
680  OPEN MID$(STR$(DRIVE),2)+":"+FILE$(I)+EX$(I) FOR INPUT AS #2
690  ADRS$=RIGHT$("000"+HEX$(CVI(INPUT$(2,2))),4)+"-"
700  ADRS$=ADRS$+RIGHT$("000"+HEX$(CVI(INPUT$(2,2))-1),4)
710  CLOSE #2
720 RETURN
```

9 - 6 - 2　ディスクエディット　リスト

```
1000 '
1010 '      DISK EDITOR
1020 '
1030 '---- INIT
1040 DEFINT A-Z
1050 WIDTH 80,25 : CONSOLE 0,25,0,0 : COLOR 0
1060 KEY 1,CHR$(253)+"HEXCODE"+CHR$(29)
1070 KEY 2,CHR$(254)+CHR$(28)+"CHARACTER"
1080 KEY 5,CHR$(255)+"END"
1090 '
1100 CLS : PRINT "(( DISKETTE EDITOR ))"
1110 INPUT "Drive    ";DRIVE : IF DRIVE=0 THEN 2270
1120 IF DSKF(DRIVE,1)=26 THEN DTYPE=3
1130 IF DSKF(DRIVE,2) THEN DTYPE=2 ELSE DTYPE=1
1140 IF DTYPE>1 THEN INPUT"Surface ";SURF
1150 INPUT "Track   ";TRACK
1160 INPUT "Sector  ";SECTOR
1170 FIELD#0,128 AS A$,128 AS B$ : GOSUB *DISK.READ
1180 '
1190 '---- DATA DISPLAY
1200 CONSOLE 0,25 : CLS
1210 PRINT "■■ DISK  EDITOR ■■   ";
1220 PRINT "Drive";DRIVE;
1230 IF DTYPE>1 THEN PRINT " Surface";SURF;
1240 PRINT " Track";TRACK;" Sector";SECTOR
1250 LOCATE 2,2
1260 FOR I=0 TO 15 : PRINT " +";HEX$(I); : NEXT
1270 PRINT SPC(6);"0123456789ABCDEF"
1280 LOCATE 3,3 : PRINT STRING$(47,"─");SPC(5);"┌";STRING$(16,"─");"┐"
1290 LOCATE 0,4
1300 FOR I=0 TO 15
1310   PRINT HEX$(I);": ";
1320   POKE &HF5DF+I*120,150
1330   FOR J=0 TO 15 : K=I*16+J
1340     IF I>7 THEN D=ASC(MID$(B$,K-127,1)) ELSE D=ASC(MID$(A$,K+1,1))
1350     PRINT RIGHT$("0"+HEX$(D),2);" ";
1360     POKE &HF5E0+I*120+J,D
1370   NEXT
1380   LOCATE 72,I+4 : PRINT"|"
1390 NEXT
1400 LOCATE 3,20 : PRINT STRING$(47,"─");SPC(5);"└";STRING$(16,"─");"┘"
1410 '
1420 '----  COMMAMD INPUT
1430 CONSOLE 22,1,0 : CLS
1440 INPUT "EDIT,SAVE,FIN (e/s/f) ";C$
1450 IF C$="" THEN 1430
1460 ON INSTR("esf",C$) GOSUB *EDITOR,*SAVER,*ENDER : GOTO 1430
1470 '
1480 '---- END
1490 *ENDER
1500  CONSOLE 0,25,0 : RETURN 1510
1510  KEY 1,"load "+CHR$(34) : KEY 2,"files " : KEY 5,"run"+CHR$(13)
1520  LOCATE 0,22,1 : CONSOLE 0,25,1
1530 END : RUN
1540 '
1550 '---- SAVE
1560 *SAVER
1570 WDRIVE=DRIVE : WSURF=SURF : WTRACK=TRACK : WSECTOR=SECTOR
1580 IF WDRIVE=0 THEN 1610
1590 INPUT "The same sector (y/n)";C$
1600 IF C$="y"OR C$="Y"THEN 1640
1610 INPUT "Drive ";WDRIVE
1620 IF DSKF(WDRIVE,2)=1 THEN INPUT "Surface,track,sector";WSURF,WTRACK,WSECTOR
1630 IF DSKF(WDRIVE,2)=0 THEN INPUT "Track,sector";WTRACK,WSECTOR
1640 '
1650 LOCATE ,,0:DIM D$(15):W=&HF5E0:H=VARPTR(W)
1660 FOR J=0 TO 15
```

```
1670    K=VARPTR(D$(J))
1680    POKE K,16:POKE K+1,PEEK(H):POKE K+2,PEEK(H+1)
1690    W=W+120
1700 NEXT
1710 A1$="" : FOR J=0 TO  7 : A1$=A1$+D$(J) : NEXT : LSET A$=A1$
1720 B1$="" : FOR J=8 TO 15 : B1$=B1$+D$(J) : NEXT : LSET B$=B1$
1730 IF DSKF(WDRIVE,1)=26 THEN DTYPE=3 : GOTO 1750
1740 IF DSKF(WDRIVE,2) THEN DTYPE=2 ELSE DTYPE=1
1750 GOSUB *DISK.WRITE : ERASE D$
1760 LOCATE ,,1 : RETURN
1770 '
1780 '---- EDIT
1790 *EDITOR
1800 LOCATE 0,22 : PRINT"Now edit mode.";
1810 LOCATE 3,4 : CONSOLE ,,1
1820 C$=INPUT$(1) : V=ASC(C$)
1830 GOSUB *KEYCLEAR
1840 X=POS(0) : Y=CSRLIN
1850 IF V=255 THEN RETURN
1860 IF V=254 THEN IF X<52 THEN LOCATE X¥3+ 55,Y : GOTO 1820 ELSE 1820
1870 IF V=253 THEN IF X>52 THEN LOCATE X*3-165,Y : GOTO 1820 ELSE 1820
1880 IF X>52 THEN 2150
1890 '     ---- HEX EDIT
1900 IF C$>="0"AND C$<="9"THEN 2030
1910 IF C$>="A"AND C$<="F"THEN 2030
1920 IF C$>="a"AND C$<="f"THEN V=V-32:C$=CHR$(V):GOTO 2030
1930 IF V=8 THEN V=29
1940 IF V=32 THEN V=28
1950 IF V>=28 AND V<=31 THEN 1970
1960 GOTO 1820
1970 ON V-27 GOTO 1980,1990,2000,2010
1980 X=X+1-(X MOD 3>0)+(X=49)*48:IF X=3 THEN 2010 ELSE 2020
1990 X=X-1+(X MOD 3=0)-(X=3)*48:IF X=49 THEN 2000 ELSE 2020
2000 Y=Y-1-(Y=4) : GOTO 2020
2010 Y=Y+1+(Y=19)
2020 LOCATE X,Y:GOTO 1820
2030 PRINT C$;
2040 D=PEEK(X¥3+Y*120+&HF3FF)
2050 K=V-48+(V>60)*7
2060 IF X MOD 3=0 THEN D=(D AND &HF)OR(K*16)
2070 IF X MOD 3=1 THEN D=(D AND &HF0)OR K
2080 POKE X¥3+Y*120+&HF3FF,D
2090 IF X MOD 3=0 THEN 1820
2100 PRINT " ";
2110 IF X<49 THEN 1820
2120 IF Y=19 THEN LOCATE 3,4 ELSE LOCATE 3,Y+1
2130 GOTO 1820
2140 '     ---- CHARACTER EDIT
2150 IF V=8 THEN V=29
2160 IF V>=28 AND V<=31 THEN 2200
2170 POKE &HF3C8+Y*120+X,V
2180 LOCATE X*3-165,Y,0 : PRINT RIGHT$("0"+HEX$(V),2):LOCATE X,Y
2190 GOTO 2210
2200 ON V-27 GOTO 2210,2220,2230,2240
2210 X=X+1+(X=71)*16 : IF X=56 THEN 2240 ELSE 2250
2220 X=X-1-(X=56)*16 : IF X<71 THEN 2250
2230 Y=Y-1-(Y=4) : GOTO 2250
2240 Y=Y+1+(Y=19)
2250 LOCATE X,Y,1 : GOTO 1820
2260 '
2270 '---- READ FROM MEMORY
2280 PRINT"Read from memory.  Input address.";:INPUT ADRS$
2290 J=VAL("&H"+ADRS$) : A1$="" : B1$=""
2300 K=VARPTR(J) : D=VARPTR(A1$)
2310 POKE D,128 : POKE D+1,PEEK(K) : POKE D+2,PEEK(K+1)
2320 J=J+128 : D=VARPTR(B1$)
2330 POKE D,128 : POKE D+1,PEEK(K) : POKE D+2,PEEK(K+1)
2340 FIELD#0,128 AS A$,128 AS B$ : LSET A$=A1$ : LSET B$=B1$
2350 GOTO 1200
2360 '
```

```
2370 '---- SUBROUTINES
2380 *DISK.READ
2390   IF DTYPE=1 THEN DUMY$=DSKI$(DRIVE,TRACK,SECTOR) : RETURN
2400   IF DTYPE=2 THEN DUMY$=DSKI$(DRIVE,SURF,TRACK,SECTOR) : RETURN
2410   IF DTYPE=3 THEN DUMY$=DSKI$(DRIVE,SURF,TRACK,SECTOR) : RETURN
2420   PRINT "DRIVE TYPE ERROR "; : BEEP : END
2430 *DISK.WRITE
2440   IF DTYPE=1 THEN DSKO$ WDRIVE,WTRACK,WSECTOR : RETURN
2450   IF DTYPE=2 THEN DSKO$ WDRIVE,WSURF,WTRACK,WSECTOR : RETURN
2460   IF DTYPE=3 THEN DSKO$ WDRIVE,WSURF,WTRACK,WSECTOR : RETURN
2470   PRINT "DRIVE TYPE ERROR "; : BEEP : END
2480 *KEYCLEAR
2490  STOP ON
2500   CKB=&H35D9 : CALL CKB 'clear queue
2510  STOP OFF
2520 RETURN
```

9－6－3 ファイルソート リスト

```
100 '
110 '   FILE  SORT
120 '
130 '---- TITLE
140 CONSOLE 0,25,1,0 : WIDTH 80,25
150 COLOR 0 : PRINT "N88-DISK BASIC ((  FILE SORT  )) "
160 PRINT
170 '
180 '---- DRIVE NUMBER INPUT
190 INPUT "DRIVE NUMBER ";DRIVE
200 IF DSKF(DRIVE,1)=26 THEN DRIVE.TYPE=3:GOTO 230 'standard
210 IF DSKF(DRIVE,1)<>16 THEN *NOT.SUPPORT
220 IF DSKF(DRIVE,2) THEN DRIVE.TYPE=2 ELSE DRIVE.TYPE=1
230 PRINT
240 '
250 '---- READ DIRECTORY to FILE$( )
260 DIM FILE$(192),ROG$(15)
270 FOR I=0 TO 15 : FIELD#0, I*16 AS DUMY$,16 AS ROG$(I) : NEXT
280 SECTOR=1 : FILE.COUNT=0
290 IF DRIVE.TYPE=2 THEN DUMY$=DSKI$(DRIVE,1,18,SECTOR)
300 IF DRIVE.TYPE=1 THEN DUMY$=DSKI$(DRIVE,18,SECTOR)
310 IF DRIVE.TYPE=3 THEN DUMY$=DSKI$(DRIVE,0,DSKF(DRIVE,5),SECTOR)
320 FOR I=0 TO 15
330   IF ASC(ROG$(I))=255 THEN *FILEEND
340   IF ASC(ROG$(I))=0   THEN LSET ROG$(I)=STRING$(16,255)
350   FILE$(FILE.COUNT)=ROG$(I)
360   FILE.COUNT=FILE.COUNT+1
370 NEXT I
380 SECTOR=SECTOR+1 : IF SECTOR<DSKF(DRIVE,10) THEN 290
390 *FILEEND
400 FILE.COUNT=FILE.COUNT-1
410 '
420 '---- SORT
430 PRINT "---- SORTING ----"
440 FOR I=0 TO FILE.COUNT-1
450 FOR J=I+1 TO FILE.COUNT
460     IF FILE$(I)>FILE$(J) THEN SWAP FILE$(I),FILE$(J)
470 NEXT J,I
480 '
490 '---- WRITE ON DISK
500 INPUT"MAY I WRITE ON DISK (y/n) ";DUMY$
510 IF DUMY$<>"y" THEN PRINT "ABORTED." : END
520 J=0 : SECTOR=1
530 FOR I=0 TO FILE.COUNT
540   LSET ROG$(J)=FILE$(I)
550   J=J+1
560   IF J=16 THEN GOSUB *DISK.OUT : J=0 : SECTOR=SECTOR+1
570 NEXT I
580 IF J=0 THEN 630
```

```
590 WHILE J<16
600   LSET ROG$(J)=STRING$(16,255) : J=J+1
610 WEND
620 GOSUB *DISK.OUT
630 PRINT "WRITE END."
640 FILES DRIVE : END
650 '
660 *NOT.SUPPORT
670  PRINT "THAT DRIVE IS NOT SUPPORTED. ": BEEP
680 END
690 '
700 *DISK.OUT
710  IF DRIVE.TYPE=2 THEN DSKO$ DRIVE,1,18,SECTOR  : RETURN
720  IF DRIVE.TYPE=1 THEN DSKO$ DRIVE,18,SECTOR  : RETURN
730  IF DRIVE.TYPE=3 THEN DSKO$ DRIVE,0,DSKF(DRIVE,5),SECTOR  : RETURN
740  PRINT "DRIVE TYPE ERROR "
750 END
```

9－6－4　ファイル・リロケーション　リスト

```
1000 '
1010 '   FILE  RELOCATION
1020 '
1030 CONSOLE 0,25,1,0 : WIDTH 80,25
1040 COLOR 0 : PRINT "N88-DISK BASIC ((  DIRECTORY RELOCATION  )) "
1050 PRINT
1060 DEFINT A-Z
1070 '
1080 '---- DRIVE NUMBER INPUT
1090 INPUT "DRIVE NUMBER ";DRIVE
1100 IF DSKF(DRIVE,1)=26 THEN DRIVE.TYPE=3 : GOTO 1130
1110 IF DSKF(DRIVE,1)<> 16 THEN *NOT.SUPPORT
1120 IF DSKF(DRIVE,2) THEN DRIVE.TYPE=2 ELSE DRIVE.TYPE=1
1130 PRINT
1140 '
1150 '---- READ DIRECTORY to FILE$( )
1160 DIM FILE$(79),ROG$(15)  'FILE# < 80
1170 FILE.END=0
1180 FOR I=0 TO 15 : FIELD#0, I*16 AS DUMY$,16 AS ROG$(I) : NEXT
1190 FOR SECTOR=1 TO 5
1200   IF DRIVE.TYPE=1 THEN DUMY$=DSKI$(DRIVE,18,SECTOR)
1210   IF DRIVE.TYPE=2 THEN DUMY$=DSKI$(DRIVE,1,18,SECTOR)
1220   IF DRIVE.TYPE=3 THEN DUMY$=DSKI$(DRIVE,0,DSKF(DRIVE,5),SECTOR)
1230   IF DRIVE.TYPE=0 THEN *NOT.SUPPORT
1240   FOR I=0 TO 15
1250     IF ASC(ROG$(I))=255 THEN FILE.END=1
1260     IF FILE.END=0 THEN FILE$(SECTOR*16-16+I)=ROG$(I)
1270     IF FILE.END=1 THEN FILE$(SECTOR*16-16+I)=STRING$(16,255)
1280   NEXT I
1290 NEXT SECTOR
1300 '
1310 '---- DISPLAY
1320 CONSOLE 0,25,1,0 : WIDTH 80,25
1330 LOCATE 0,0 : PRINT "DIRECTORY RELOCATION"
1340 FOR SECTOR=1 TO 5
1350   LOCATE (SECTOR-1)*16,2 : PRINT " SECTOR  ";SECTOR
1360 NEXT
1370 LOCATE 0,3 : PRINT STRING$(79,"-")
1380 FOR I=0 TO 15
1390   FOR SECTOR=1 TO 5
1400     J=SECTOR-1
1410     LOCATE J*16,I+4
1420     FILE$=FILE$(J*16+I)
1430     GOSUB *MAKE.FILE.NAME
1440     PRINT " ";FILE$
1450   NEXT SECTOR
1460 NEXT I
1470 LOCATE 0,20 : PRINT STRING$(79,"-")
1480 '
```

```
1490 '---- PREPARATION FOR EDIT
1500 CONSOLE ,,0
1510 KEY 1,CHR$(255)+"MARK"
1520 KEY 2,CHR$(&HFD)+"SWAP"
1530 KEY 3,CHR$(&HFC)+"RENAME"
1540 KEY 4,""
1550 KEY 5,CHR$(&HFE)+"END"
1560 CONSOLE ,,1
1570 COMMAND$=CHR$(31)+CHR$(30)+CHR$(29)+CHR$(28)
1580 COMMAND$=COMMAND$+CHR$(255)+CHR$(254)+CHR$(253)+CHR$(252)
1590 MARK=-1 : CURSOR=0
1600 '
1610 '---- EDIT
1620 X=(CURSOR ¥ 16) *16 : Y=(CURSOR MOD 16) +4
1630 CCOLOR=4 : IF CURSOR=MARK THEN CCOLOR=6
1640 COLOR@ (X,Y)-(X+11,Y),CCOLOR
1650 LOCATE ,,0  : CKB=&H35D9 : CALL CKB
1660 P=INSTR(COMMAND$,INPUT$(1)) : IF P=0 THEN 1660
1670 ON P GOSUB *DN,*UP,*LF,*RI,*MARK,*EXIT,*SWAP.,*RENAME
1680 CCOLOR=0 : IF CURSOR=MARK THEN CCOLOR=2
1690 COLOR@ (X,Y)-(X+11,Y),CCOLOR
1700 CURSOR=NEWCURSOR
1710 GOTO 1620
1720 '
1730 *DN
1740   NEWCURSOR=CURSOR+1
1750   IF NEWCURSOR>=80 THEN NEWCURSOR=0
1760   RETURN
1770 *UP
1780   NEWCURSOR=CURSOR-1
1790   IF NEWCURSOR<0 THEN NEWCURSOR=79
1800   RETURN
1810 *LF
1820   NEWCURSOR=CURSOR-16
1830   IF NEWCURSOR<0 THEN NEWCURSOR=NEWCURSOR+80
1840   RETURN
1850 *RI
1860   NEWCURSOR=CURSOR+16
1870   IF NEWCURSOR>79 THEN NEWCURSOR=NEWCURSOR-80
1880   RETURN
1890 *MARK
1900   CKB=&H35D9:CALL CKB 'clear queue
1910   MX=(MARK ¥ 16) *16 : MY=(MARK MOD 16) +4
1920   IF MARK>-1 THEN COLOR@ (MX,MY)-(MX+11,MY),0
1930   MARK=CURSOR
1940   IF ASC(FILE$(MARK))=255 THEN BEEP : MARK=-1
1950   RETURN
1960 *EXIT
1970   CKB=&H35D9:CALL CKB 'clear queue
1980   MX=(MARK ¥ 16) *16 : MY=(MARK MOD 16) +4
1990   COLOR @(MX,MY)-(MX+11,MY),0
2000   MX=(CURSOR ¥ 16) *16 : MY=(CURSOR MOD 16) +4
2010   COLOR @(MX,MY)-(MX+11,MY),0
2020   CONSOLE ,,0
2030   KEY 1,"load "+CHR$(34)
2040   KEY 2,"files "
2050   KEY 3,"go to "
2060   KEY 4,"list "
2070   KEY 5,"run"+CHR$(13)
2080   CONSOLE ,,1
2090   LOCATE ,,1
2100   GOTO *WRITE.ON.DISK
2110 *SWAP.
2120   CKB=&H35D9:CALL CKB 'clear queue
2130   IF MARK<0 THEN BEEP : RETURN
2140   IF ASC(FILE$(CURSOR))=255 THEN BEEP : RETURN
2150   SWAP FILE$(MARK),FILE$(CURSOR)
2160   MX=(MARK ¥ 16) *16 : MY=(MARK MOD 16) +4
2170   FILE$=FILE$(MARK) : GOSUB *MAKE.FILE.NAME
2180   LOCATE MX,MY : COLOR 2 : PRINT " ";FILE$
```

```
2190   MX=(CURSOR ¥ 16) *16 : MY=(CURSOR MOD 16) +4
2200   FILE$=FILE$(CURSOR) : GOSUB *MAKE.FILE.NAME
2210   LOCATE MX,MY : COLOR 2 : PRINT " ";FILE$
2220   COLOR 0
2230   RETURN
2240 *RENAME
2250   CKB=&H35D9:CALL CKB 'clear queue
2260   P=ASC(FILE$(CURSOR)) : IF (P=0) OR (P=255) THEN BEEP : RETURN
2270   CONSOLE 22,23 : CLS : LOCATE,,1
2280   INPUT "NEW FILE NAME";FILE$ : IF FILE$="" THEN 2450
2290   IF (ASC(FILE$)=0) OR (ASC(FILE$)=255) THEN BEEP : GOTO 2280
2300   P=INSTR(FILE$,".")' : NFILE$=STRING$(9," ")
2310   IF P=1 THEN BEEP : GOTO 2280
2320   IF P=0 AND LEN(FILE$)>6 THEN FILE$=LEFT$(FILE$,6)+"."+MID$(FILE$,7,3)
2330   IF P=0 THEN P=7
2340   MID$(NFILE$,1,6)=LEFT$(FILE$,P-1)
2350   MID$(NFILE$,7,3)=MID$(FILE$,P+1)
2360   FOR P=0 TO 79
2370     IF ASC(FILE$(P))=255 THEN P=79 : GOTO 2400
2380     IF LEFT$(FILE$(P),9)<>NFILE$ THEN 2400
2390     PRINT "EXIST FILE NAME.";CHR$(7) : GOTO 2280
2400   NEXT
2410   MID$(FILE$(CURSOR),1,9)=NFILE$
2420   MX=(CURSOR ¥ 16) *16 : MY=(CURSOR MOD 16) +4
2430   FILE$=FILE$(CURSOR) : GOSUB *MAKE.FILE.NAME
2440   LOCATE MX,MY : COLOR 2 : PRINT " ";FILE$
2450   COLOR 0 : LOCATE,,0
2460   CLS : CONSOLE 0,25
2470   RETURN
2480 '
2490 '---- WRITE RELOCATED DIRECTORY AND END
2500 *WRITE.ON.DISK
2510 LOCATE 0,22:INPUT "WRITE ON DISK (y/n/r) ";DUMY$
2520 IF DUMY$="y" THEN 2550
2430   FILE$=FILE$(CURSOR) : GOSUB *MAKE.FILE.NAME
2440   LOCATE MX,MY : COLOR 2 : PRINT " ";FILE$
2450   COLOR 0 : LOCATE,,0
2460   CLS : CONSOLE 0,25
2470   RETURN
2480 '
2490 '---- WRITE RELOCATED DIRECTORY AND END
2500 *WRITE.ON.DISK
2510 LOCATE 0,22:INPUT "WRITE ON DISK (y/n/r) ";DUMY$
2520 IF DUMY$="y" THEN 2550
2530 IF DUMY$="r" THEN LOCATE 0,22 : PRINT SPC(30); : RETURN 1490 'EDIT
2540 PRINT "ABORTED." : END
2550 FOR SECTOR=1 TO 5
2560   FOR J=0 TO 15
2570     LSET ROG$(J)=FILE$(SECTOR*16-16+J)
2580   NEXT J
2590   GOSUB *DISK.OUT
2600 NEXT SECTOR
2610 FILES DRIVE
2620 END
2630 '
2640 '---- MAKE FILENAME ROUTINE
2650 *MAKE.FILE.NAME
2660   IF LEFT$(FILE$,9)=STRING$(9,255) THEN FILENAME$="( Empty ) " :GOTO 2690
2670   FILENAME$=LEFT$(FILE$,6)+" "+MID$(FILE$,7,3)
2680   IF ASC(FILENAME$)=0 THEN FILENAME$="( Killed )"
2690   FILE$=FILENAME$
2700 RETURN
2710 '
2720 '---- DSKO$ ROUTINE
2730 *DISK.OUT
2740   IF DRIVE.TYPE=1 THEN DSKO$ DRIVE,18,SECTOR : RETURN
2750   IF DRIVE.TYPE=2 THEN DSKO$ DRIVE,1,18,SECTOR : RETURN
2760   IF DRIVE.TYPE=3 THEN DSKO$ DRIVE,0,DSKF(DRIVE,5),SECTOR : RETURN
2770   PRINT "DISK TYPE ERROR "
2780 END
2790 '
```

```
2800 '---- NOT SUPPORTED ERROR
2810 *NOT.SUPPORT
2820   PRINT "THAT DRIVE IS NOT SUPPORTED. "
2830 END
```

9-6-5 選択的ファイル転送 リスト

```
1000 '
1010 '    Selective File Transfer  (1984.6.9)
1020 '
1030 DEFINT A-Z
1040 ON STOP GOSUB *ENDE : STOP ON
1050 ON ERROR GOTO *ON.ERROR
1060 '
1070 '---- Arrays & Constants
1080 DIM FILE$(100), ATR(100) : FCT = 0
1090 DIM DIR$(15) : FOR I=0 TO 15 : FIELD#0,I*16 AS DM$,16 AS DIR$(I) : NEXT
1100 FIELD#1, 128 AS A1$, 128 AS B1$
1110 FIELD#2, 128 AS A2$, 128 AS B2$
1120 MAX.DR = PEEK(&HEC7D)
1130 DEF FNDTYP(DR) = PEEK(&HEF63+DR)
1140 IOBUF0 = VARPTR(#0)+9
1150 '
1160 '---- Enter drive#
1170 WIDTH 80, 25 : CONSOLE 0,25,1,0 : LOCATE ,,1
1180 LOCATE 10,8  : PRINT "■■■■ Selective File Transfer ■■■■" : PRINT
1190 '
1200 LOCATE 12,10 : PRINT ">>Source drive number: ";
1210 LINE INPUT A$ : S.DR=VAL(A$)
1220 IF S.DR<1 OR MAX.DR<S.DR THEN GOSUB *BAD.DRIVE : GOTO 1200
1230 '
1240 LOCATE 12,12 : PRINT ">>Destination drive number: ";
1250 LINE INPUT A$ : D.DR=VAL(A$)
1260 IF D.DR<1 OR MAX.DR<D.DR OR D.DR=S.DR THEN GOSUB *BAD.DRIVE : GOTO 1240
1270 '
1280 '---- Set drive constans by S.DR, D.DR
1290 S.MAX.DIRSC = DSKF(S.DR, 8)
1300 '
1310 D.FREE = DSKF(D.DR)
1320 D.MAX.DIRSC = DSKF(D.DR, 8)
1330 '
1340 '---- Diaplay Source directory
1350 CONSOLE 0,25,0,0 : WIDTH 80,25 : LOCATE 0,0,0
1360 F$= "   Drive# -> Drive#(FREE   )"
1370 PRINT "■■ Selective File Transfer ■■ ";
1380 PRINT USING F$; S.DR, D.DR
1390 GOSUB *DISP.FREE
1400 PRINT STRING$(78, "-")
1410 FOR SECTOR=1 TO S.MAX.DIRSC
1420   DR = S.DR : GOSUB *DSKI.DIR
1430   FOR I=0 TO 15
1440     IF ASC(DIR$(I))=0 THEN 1580
1450     IF ASC(DIR$(I))=255 THEN *END.OF.DIR
1460     '
1470     LOCATE (FCT MOD 5)*16+2, FCT¥5 + 2
1480     FILE$(FCT) = LEFT$(DIR$(I), 9)
1490     ATR(FCT)=ASC(MID$(DIR$(I),10))
1500     FILE$ = FILE$(FCT) : ATR = ATR(FCT) : FCT = FCT + 1
1510     PRINT LEFT$(FILE$,6);
1520     IF ATR AND &H80 THEN PRINT "."; : GOTO 1550
1530     IF ATR AND 1    THEN PRINT "*"; : GOTO 1550
1540     PRINT " ";
1550     PRINT MID$(FILE$,7,3);
1560     IF FCT >= 90 THEN *END.OF.DIR
1570     '
1580   NEXT
1590  NEXT
1600 *END.OF.DIR
1610 LAST.LINE = (FCT-1)¥5 + 3
```

```
1620 LOCATE 0,LAST.LINE : PRINT STRING$(78, "-")
1630 PRINT "        Cursor keys to move, Space to select/deselect,";
1640 PRINT " Return key to end  "
1650 LOCATE 2,LAST.LINE+3 : PRINT ">> Select the files to transfer."
1660 COLOR 2
1670 LOCATE 2,LAST.LINE+4 : PRINT "  ________________________________________  ";
1680 COLOR 0
1690 '
1700 '---- Select xfer files
1710 CFN = 0
1720 '
1730 *KEY.IN
1740  PX = (CFN MOD 5)*16+2 : PY = CFN¥5 + 2
1750  COLOR@(PX,PY)-(PX+9,PY),4
1760  KIN = ASC(INPUT$(1))
1770  CKB = &H35D9 : CALL CKB    'clear queue
1780  IF KIN = 27 THEN *ENDE
1790  IF KIN = 32 THEN GOSUB *MARK : GOTO *KEY.IN
1800  IF KIN = 13 THEN *DO.XFER
1810  IF KIN<28 OR KIN>31 THEN *KEY.IN
1820  COLOR@(PX,PY)-(PX+9,PY),0
1830  ON KIN-27 GOSUB *MLEFT, *MRIGHT, *MUP, *MDOWN
1840 GOTO *KEY.IN
1850 '
1860 *MLEFT
1870  CFN = CFN + 1 : IF CFN>=FCT THEN CFN=0
1880 RETURN
1890 *MRIGHT
1900  CFN = CFN - 1 : IF CFN<0 THEN CFN = FCT - 1
1910 RETURN
1920 *MUP
1930  IF CFN>4 THEN CFN = CFN - 5
1940 RETURN
1950 *MDOWN
1960  IF CFN<FCT-5 THEN CFN = CFN + 5
1970 RETURN
1980 *MARK
1990  ATR(CFN) = ATR(CFN) XOR &H100
2000  IF (ATR(CFN) AND &H100)=0 THEN 2050
2010    GOSUB *CALC.LOF : D.FRE = D.FRE - LOFILE : GOSUB *DISP.FREE
2020    LOCATE PX-1,  PY : PRINT "["
2030    LOCATE PX+10, PY : PRINT "]"
2040 RETURN
2050    GOSUB *CALC.LOF : D.FRE = D.FRE + LOFILE : GOSUB *DISP.FREE
2060    LOCATE PX-1,  PY : PRINT " "
2070    LOCATE PX+10, PY : PRINT " "
2080 RETURN
2090 '
2100 '---- Excute file xfer
2110 *DO.XFER
2120  COLOR@(PX,PY)-(PX+9,PY),0
2130  GOSUB *CLEAR.CMDL
2140  FOR CFN = 0 TO FCT-1
2150    IF (ATR(CFN) AND &H100) = 0 THEN 2250
2160    FILE$ = FILE$(CFN) : ATR = ATR(CFN) AND &HFF
2170    LOCATE 0, LAST.LINE + 2
2180    PRINT "Transfering: [";
2190    PRINT LEFT$(FILE$,6);
2200    IF ATR AND &H80 THEN PRINT "."; : GOTO 2230
2210    IF ATR AND 1    THEN PRINT "*"; : GOTO 2230
2220    PRINT " ";
2230    PRINT MID$(FILE$,7,3);"]";
2240    GOSUB *XFER.A.FILE
2250  NEXT
2260 '
2270 GOTO *ENDE
2280 '
2290 '---- Transfer a file "FILE$"
2300 *XFER.A.FILE
2310 OPEN HEX$(S.DR)+":"+FILE$ AS #1
2320 OPEN HEX$(D.DR)+":"+FILE$ AS #2
```

```
2330 '
2340 FOR I=LOF(1) TO 1 STEP -1
2350   GET#1, I
2360   LSET A2$=A1$ : LSET B2$=B1$
2370   PUT#2, I
2380 NEXT
2390 '
2400 CLOSE
2410 '
2420 FOR SECTOR=1 TO D.MAX.DIRSC
2430   DR = D.DR : GOSUB *DSKI.DIR
2440   FOR I=0 TO 15
2450    IF ASC(DIR$(I))=0 THEN 2480      :' killed file
2460    IF ASC(DIR$(I))=255 THEN *ERR.NOFILE
2470    IF LEFT$(DIR$(I),9)=FILE$ THEN 2520
2480   NEXT
2490 NEXT
2500 GOTO *ERR.NOFILE
2510 '
2520 MID$(DIR$(I),10,1)=CHR$(ATR)
2530 DR = D.DR : GOSUB *DSKO.DIR
2540 '
2550 RETURN
2560 '
2570 '---- dski directory sector of drive DR
2580 *DSKI.DIR
2590  DTYPE = FNDTYP(DR)
2600  IF DTYPE = 0 THEN DM$ = DSKI$(DR, 0, 35, SECTOR)
2610  IF DTYPE = 1 THEN DM$ = DSKI$(DR, 1, 18, SECTOR)
2620  IF DTYPE = 2 THEN DM$ = DSKI$(DR, 18, SECTOR)
2630  IF DTYPE = 3 THEN DM$ = DSKI$(DR, 1, 18, SECTOR)
2640 RETURN
2650 '
2660 '---- dsko directory sector of drive DR
2670 *DSKO.DIR
2680  DTYPE = FNDTYP(DR)
2690  IF DTYPE = 0 THEN DSKO$ DR, 0, 35, SECTOR
2700  IF DTYPE = 1 THEN DSKO$ DR, 1, 18, SECTOR
2710  IF DTYPE = 2 THEN DSKO$ DR, 18, SECTOR
2720  IF DTYPE = 3 THEN DSKO$ DR, 1, 18, SECTOR
2730 RETURN
2740 '
2750 '---- Error section
2760 *BAD.DRIVE
2770  LOCATE 12,CSRLIN-1 : PRINT "Bad drive number";SPC(20) : BEEP
2780 RETURN
2790 '
2800 *ERR.NOFILE
2810 PRINT "Err: no file" : BEEP : END
2820 '
2830 *ON.ERROR
2840  LOCATE ,,1 : ON ERROR GOTO 0 : END
2850 '
2860 '---- Clear command lines
2870 *CLEAR.CMDL
2880  CONSOLE LAST.LINE+1 , LAST.LINE+5 : CLS
2890  CONSOLE 0,25
2900 RETURN
2910 '
2920 '---- Calculate LOF of FILE$(CFN)
2930 *CALC.LOF
2940  OPEN HEX$(S.DR)+":"+FILE$(CFN) FOR INPUT AS#1
2950  LOFILE = (LOF(1)+7) ¥ DSKF(S.DR, 6)
2960  IF LOFILE = 0 THEN LOFILE = 1
2970  CLOSE
2980 RETURN
2990 '
3000 '---- Display free of D.DR
3010 *DISP.FREE
3020  LOCATE 54,0  : PRINT USING"###"; D.FRE
3030 RETURN
```

```
3040 '
3050 '---- STOP & ESC
3060 *ENDE
3070  STOP OFF
3080  GOSUB *CLEAR.CMDL
3090  COLOR 0
3100  CONSOLE 0,25,1
3110  LOCATE 0,LAST.LINE+2,1
3120 END
```

第10章　サブシステム

PC-8801mkⅡは、最初から5インチフロッピーディスクを内蔵するように設計されています。PC-8801とのコンパチビリティーから、PC-8801用ディスクドライブであるPC-80S31と同じシステムが採用されており、従来のディスク用ソフトがそのまま使えるようになっています。

このディスク制御用のシステムは、システム専用のCPUを持ち、PC-8801mkⅡ本体のCPUから独立して動いています。第10章では、このサブシステムの使い方を中心に説明していきます。

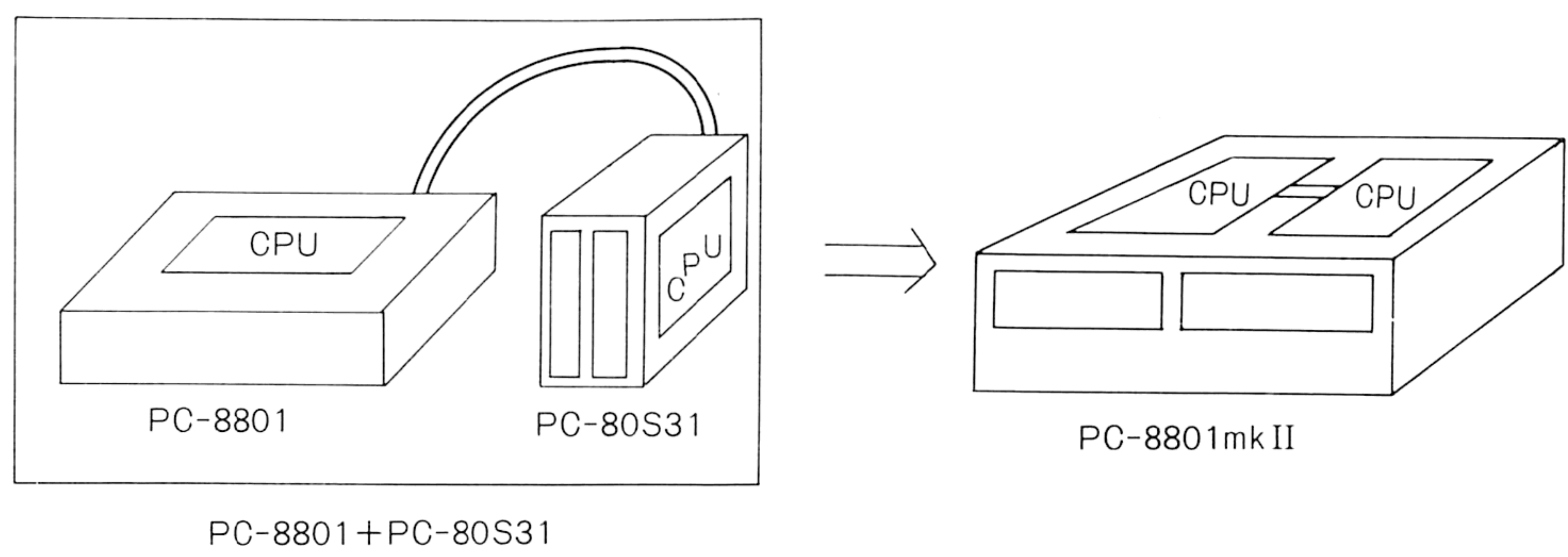



図10-A

10-1　メモリマップ

PC-8801mkⅡのサブシステムには、メインシステムと同じZ-80AコンパチブルなCPUが使われ、ROM・RAM共にメインシステムとは別に用意されています。図10-1-Aが、そのメモリマップです。ディスク制御用のプログラムが入ったROMが2Kバイト、RAMが16Kバイトあります。RAM16Kバイトのうち、サブシステムが使っているのは、ディスケットへ書き込むデータを置くための4Kバイト(4000H～4FFFH)、ディスケットから読み込んだデータを置くための4Kバイト(5000H～5FFFH)、ワークエリア及びスタックエリアの256バイト(7F00H～7FFFH)だけです。6000H～7EFFHのRAMエリアは、ユーザーが勝手に使っても、サブシステムには何の影響も与えません。

アドレス	内容	
FFFFH～8000H	未　使　用	
8000H～7F00H	ワーク・エリア及びスタック・エリア	RAM 16Kバイト
7F00H～6000H	空　き　領　域	
6000H～5000H	リード・バッファ	
5000H～4000H	ライト・バッファ	
4000H～800H	未　使　用	
800H～0H	DISK制御用プログラム	ROM 2Kバイト

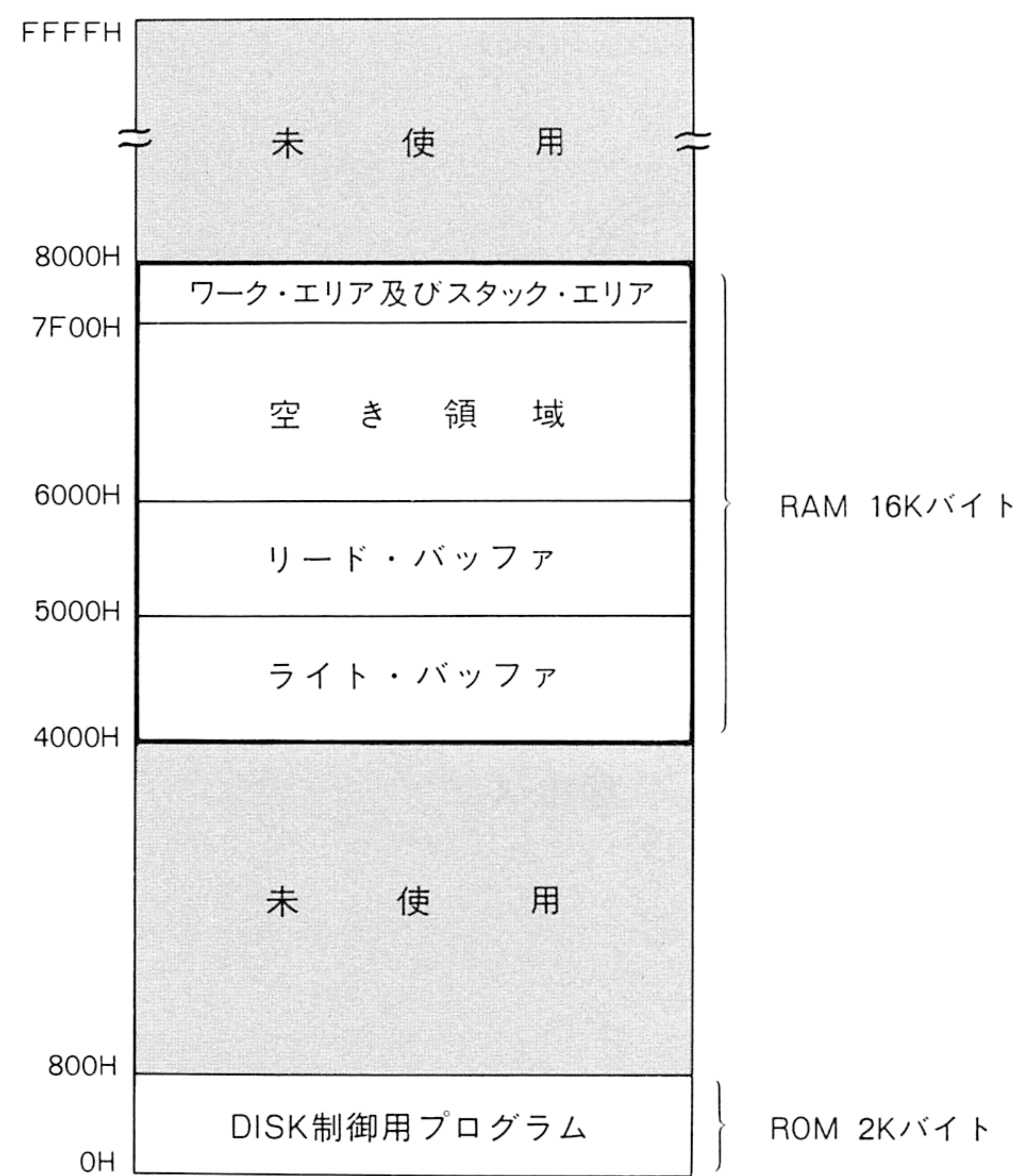



図10-1-A　メモリマップ

10-2　メインシステムとのインタフェース

10-2-1 ハンドシェイクのアルゴリズム

メインシステムとサブシステム間のデータのやり取りは、汎用パラレルインタフェースLSI・μPD-8255を用いた3線式のハンドシェイクによって行なわれます。ハンドシェイクとは、信号の交換によってデータのやり取りを行なう方法のことです。コンピュータ以外の題材で、例を上げてみましょう。

「8月の始め、私は息子に、東京の祖母の所へ遊びに行く約束をさせられました。お盆に行く予定にしていたのですが、彼はそれまで待てないらしく、何度もせがむので、一人で行かせることにしたのです。祖母に電話で了解を得て(①)、その週の日曜日、飛行機に乗せてやりました。

飛行機が離陸するのを見届けてから祖母の家へ電話をし、今出たことを告げ(②)、家に帰りました。数時間後祖母より無事着いたとの連絡があり(③)....」

福岡にいる私（メインシステム）と東京にいる祖母(サブシステム)との間で、息子(データ)をハンドシェイクしたことになります。

ハンドシェイクとは、大事なデータ(息子)を確実にやり取りするための手順と考えて下さい。

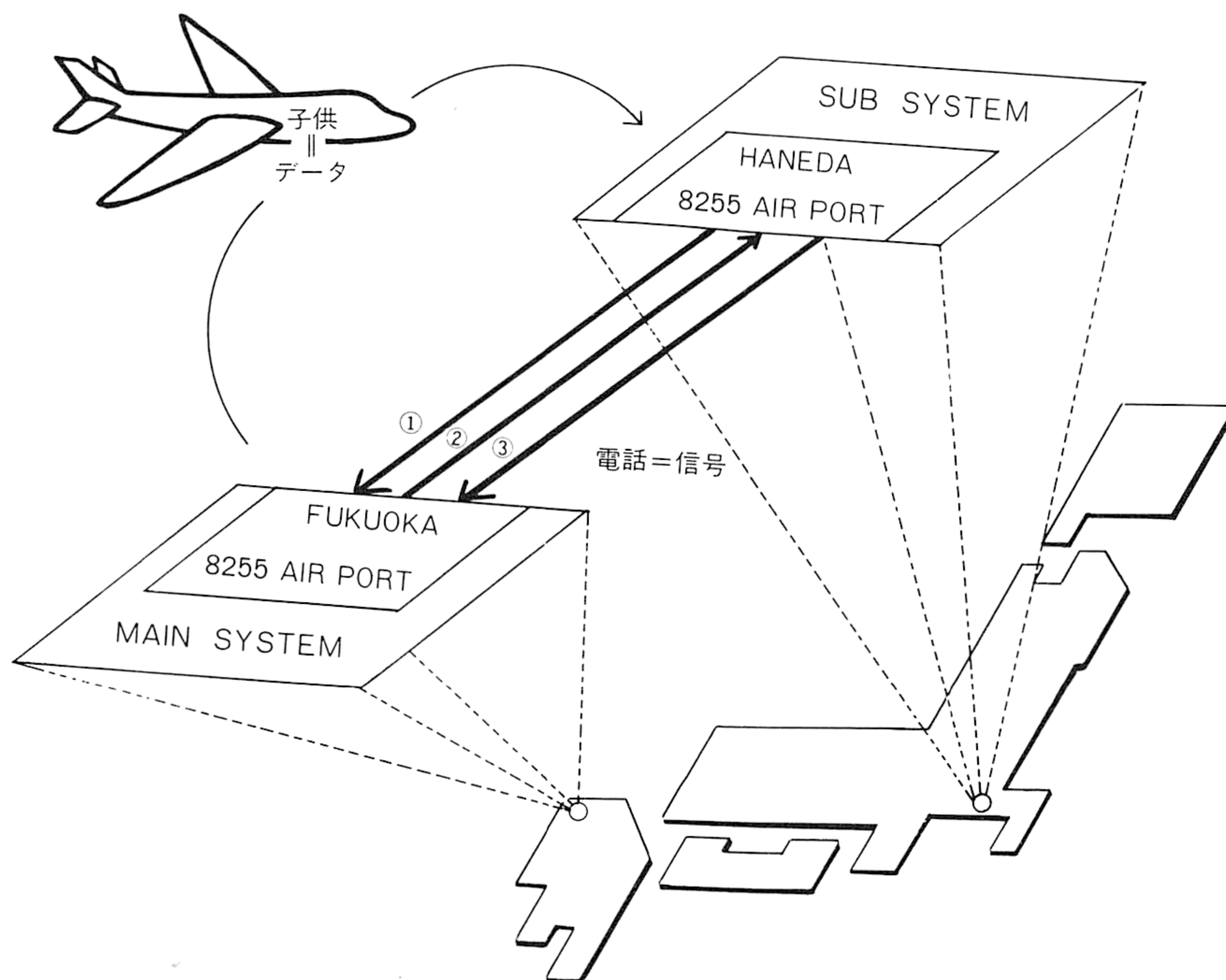



図10-2-A

サブシステムが「準備ができたので、データを送っても良い」というので(①)
メインシステムはデータを送り、そのことをサブシステムに伝えます。(②)
サブシステムはデータを受け取ると、そのことをメインシステムに知らせます。(③)

こうすればデータは、間違いなくメインシステムからサブシステムへ送られることになります。そして①，②，③3つの電話が、それぞれ3つの役割を持った信号ということになるのです。

信号線は電圧の高い・低いで1，0を示すことができます。信号が0から1になったとき、ある意味を持つように予め決めておきます。ですから信号は普通、0の状態にしておかなければなりません。でないと、誤った意味を伝えることになります。

3線式ハンドシェイクで使われる3つの信号は、RFD，DAV，DACと呼ばれ、それぞれが前の例の①，②，③にあたる意味を持っています。

(1) RFD (Ready For Data)

データを受け取る側が、RFDを1にすることで、データを送る側に、データを受け取る準備ができていることを知らせます。

(2) DAV (Data Valid)

データを送る側が、DAVを1にすることで、データを送ったことを受信側に知らせます。

(3) DAC (Data Accepted)

データを受け取る側が、DACを1にすることで、送信側にデータを受け取ったことを知らせます。

図10-2-Bを見て下さい。メインシステムからサブシステムへデータを送るハンドシェイクのフローチャートです。注意しなければならないのは、使った信号は必ず0に戻しておくということです。図10-2-Cは、PC-8801mkⅡで使われているハンドシェイクルーチンに基づいたフローチャートです。RFDを0に戻すときのチェックを省略し、DAC＝1のチェックの所で兼用させている以外はまったく同じです。

データを送る側
メイン・システム
1ではない
RFDは1か
1だ
データを送る
DAVを1にする
1ではない
DACは1か
1だ
0ではない
RFDは0か
0だ
DAVを0に戻す
0ではない
DACは0か
0だ
終了

(データを送ってもいいぞ!)
RFD=1
(データを送ったぞ)
DAV=1
データ
(データを受けとったぞ)
DAC=1
RFD=0
DAV=0
DAC=0
※ 次のハンドシェイクのために各信号(RFD, DAV, DAC)を0に戻す

データを受けとる側
サブ・システム
RFDを1にする
1ではない
DAVは1か
1だ
データを受けとる
DACを1にする
RFDを0に戻す
0ではない
DAVは0か
0だ
DACを0に戻す
終了
※

図10-2-B

送り側
メイン・システム
1ではない
RFDは1か
1だ
データを送る
DAVを1にする
1ではない
DACは1か
1だ
DAVを0にする
0ではない
DACは0か
0だ
終了

(データを送ってもいいぞ)
RFD
(データを送ったぞ)
DAV
データ
(データを受けとった)
DAC
DAV
DAC

受け取り側
サブ・システム
RFDを1にする
1ではない
DAVは1か
1だ
RFDを0にする
データを受けとる
DACを1にする
0ではない
DAVは0か
0だ
DACを0にする
終了

図10-2-C

10-2-2　高速ハンドシェイクのアルゴリズム

データのハンドシェイクの欠点は、信号のチェックの回数が多いために、データの転送に時間がかかるということです。

図10-2-Bでは、1バイトのデータ転送に6回も信号のチェックが行なわれています。もし、これが3回に減れば、速度は2倍になるはずです。1バイトの転送に使う信号のチェック回数を減らすことを考えてみましょう。図10-2-Cでは、各信号が0から1になるときに意味を持たせてハンドシェイクを行ない、信号を0→1にしたのと同じ手順で1→0に戻しています。

では、1から0になるときも意味を持たせてデータのやり取りを行なえばどうでしょうか。そうすれば1回のハンドシェイクで2バイトのデータを送受信できることになります。

このように考えて高速化したものが図10-2-Dです。PC-8801mkⅡでも、データの転送のとき、高速化したハンドシェイク(図10-2-E)を使っています。

では、もっと速くするにはどうしたらよいでしょうか。図10-2-Fを見て下さい。転送するバイト数が決まっていれば、このようなこともできます。これで20～30%速くなります。

システムソフトのアドベンチャーゲーム、“ミオのミステリーアドベンチャー”で使ったのがこの手法です。

このゲームは、システムソフト初のアドベンチャーゲームで、高速ハンドシェイクと画面データの圧縮により、速いものでは1～2秒で絵を表示できるようになっています。48Kバイトあるグラフィック画面のデータを15Kバイト程に圧縮すると、サブシステムのRAM上にすべておくことができます。それから高速転送してグラフィック画面に展開してやると、1～2秒で表示したように見えます。ただし、絵が複雑すぎて圧縮率が下がると、極端に表示速度が遅くなります。

この欠点を、ディスクの制御プログラムを独自に作ることで解決したのが“ミコとアケミのジャングルアドベンチャー”です。このアドベンチャーゲームはディスケット2枚に約80種類の絵が入っており、ディスクのアクセスランプがついてから2～5秒後には絵の表示が完了するようになりました。

データを送る側

メイン・システム
RFDは1か
1ではない
1だ
1番目の
データを送る
DAVを1にする
DACは1か
1ではない
1だ
RFDは0か
0ではない
0だ
2番目の
データを送る
DAVを0にする
DACは0か
0ではない
0だ
終　了

(データを送ってもいぞ❗)
RFD=1
(データを送ったぞ)
DAV=1
データ
(データを受けとったぞ)
DAC=1
(データを送ってもいぞ❗)
RFD=0
(データを送ったぞ)
DAV=0
データ
(データを受けとったぞ)
DAC=0

データを受けとる側

サブ・システム
RFDを1にする
DAVは1か
1ではない
1だ
データを受けとる
DACを1にする
RFDを0にする
DAVは0か
0ではない
0だ
データを受けとる
DACを0にする
終　了

図10-2-D

送り側

メイン・システム
RFDは1か
1ではない
1だ
1番目の
データを送る
DAVを1にする
DACは1か
1ではない
1だ
2番目の
データを送る
DAVを0にする
DACは0か
0ではない
0だ
終　了

(データを送って
もいいぞ❗)
RFD
(データを送ったぞ)
DAV
データ
(データを受けとった)
DAC
DAV
データ
DAC

受け取り側

サブ・システム
RFDを1にする
DAVは1か
1ではない
1だ
RFDを0にする
データを
受けとる
DACを1にする
DAVは0か
0ではない
0だ
データを
受けとる
DACを0にする
終　了

図10-2-E

データを送る側
メイン・システム
1ではない
RFDは1か
1だ
データを送る
DAVを1にする
1ではない
DACは1か
1だ
データを送る
DAVを0にする
0ではない
DACは0か
0だ
0ではない
RFDは0か
0だ
終了

データを受けとる側
サブ・システム
RFDを1にする
1ではない
DAVは1か
1だ
データを受けとる
DACを1にする
0ではない
DAVは0か
0だ
データを受けとる
DACを0にする
RFDを0に戻す
終了

（データを送ってもいいぞ！）
RFD＝1
（データを送ったぞ）
DAV＝1
データ
（データを受けとったぞ）
DAC＝1
（データを送ったぞ）
DAV＝0
データ
（データを受けとったぞ）
DAC＝0
RFD＝0

転送バイト数／2回くり返す

図10－2－F

さて、ここまではハンドシェイクのチェック回数を減らすことで速度を上げていきました。次は、１回のハンドシェイで送れるデータの数を増やすことを考えてみましょう。これは8255というLSIでのみ可能な手段で、他のLSIによるハンドシェイクでは使えません。

8255には3つの8ビットポートがあり、それぞれをA，B，Cといいます。

ポートAをデータの入力、ポートBをデータの出力、ポートCをハンドシェイクの信号RFD,DAV,DAC用と決められています。この状態ではデータは1バイトしか送れません。

そこで、データを送る場合はポートAもデータの出力に使い、データを受け取る場合はポートBもデータの入力に使うとしたらどうでしょうか？

図10-２-Gを見て下さい。このように、一度に２バイトのデータを送受信できます。つまり、速度は2倍になるわけです。

メイン・システム
サブ・システム
ポートBでデータを送る
DAVを1にする
（データを送ったぞ）
DAV＝1
1ではない
DAVは1か
1だ
データ
データを受けとる

出力ポートが1つのとき（ポートBのみ）

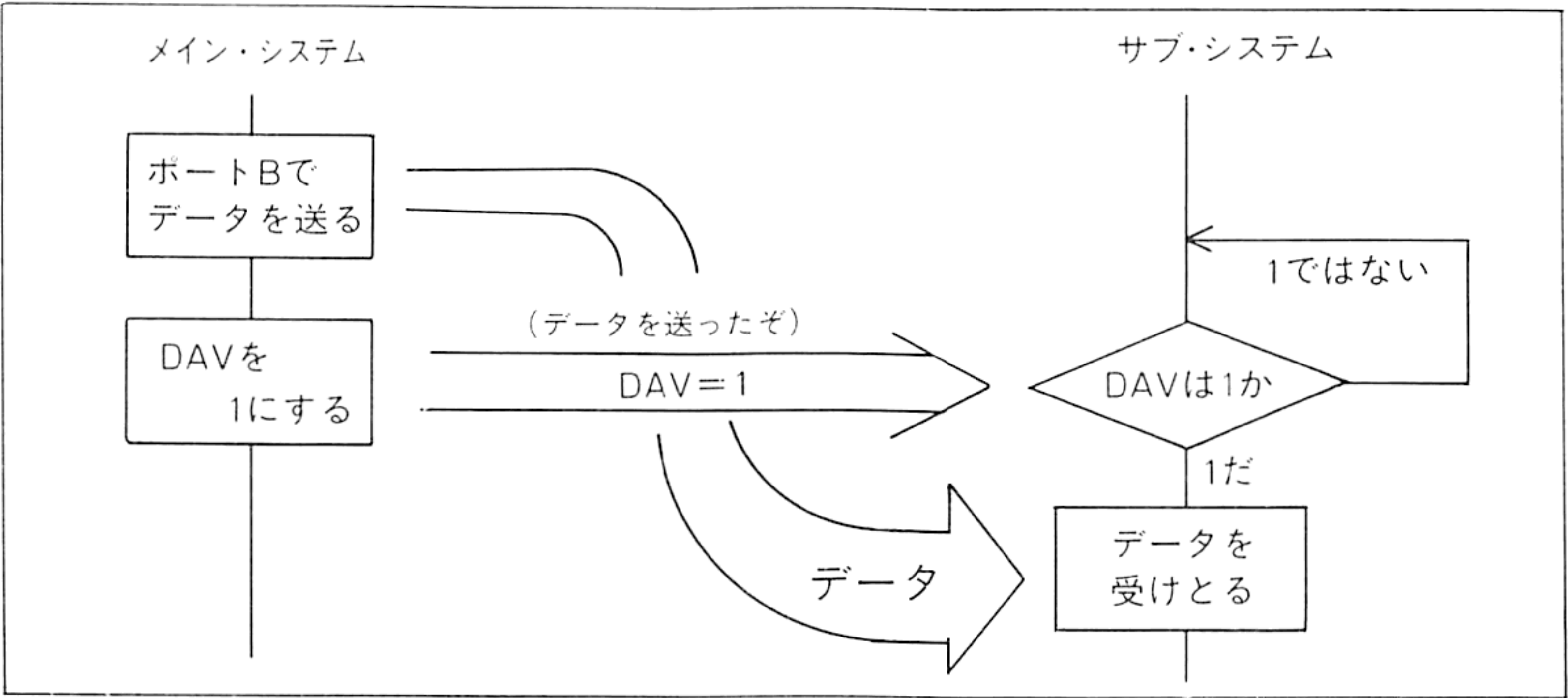
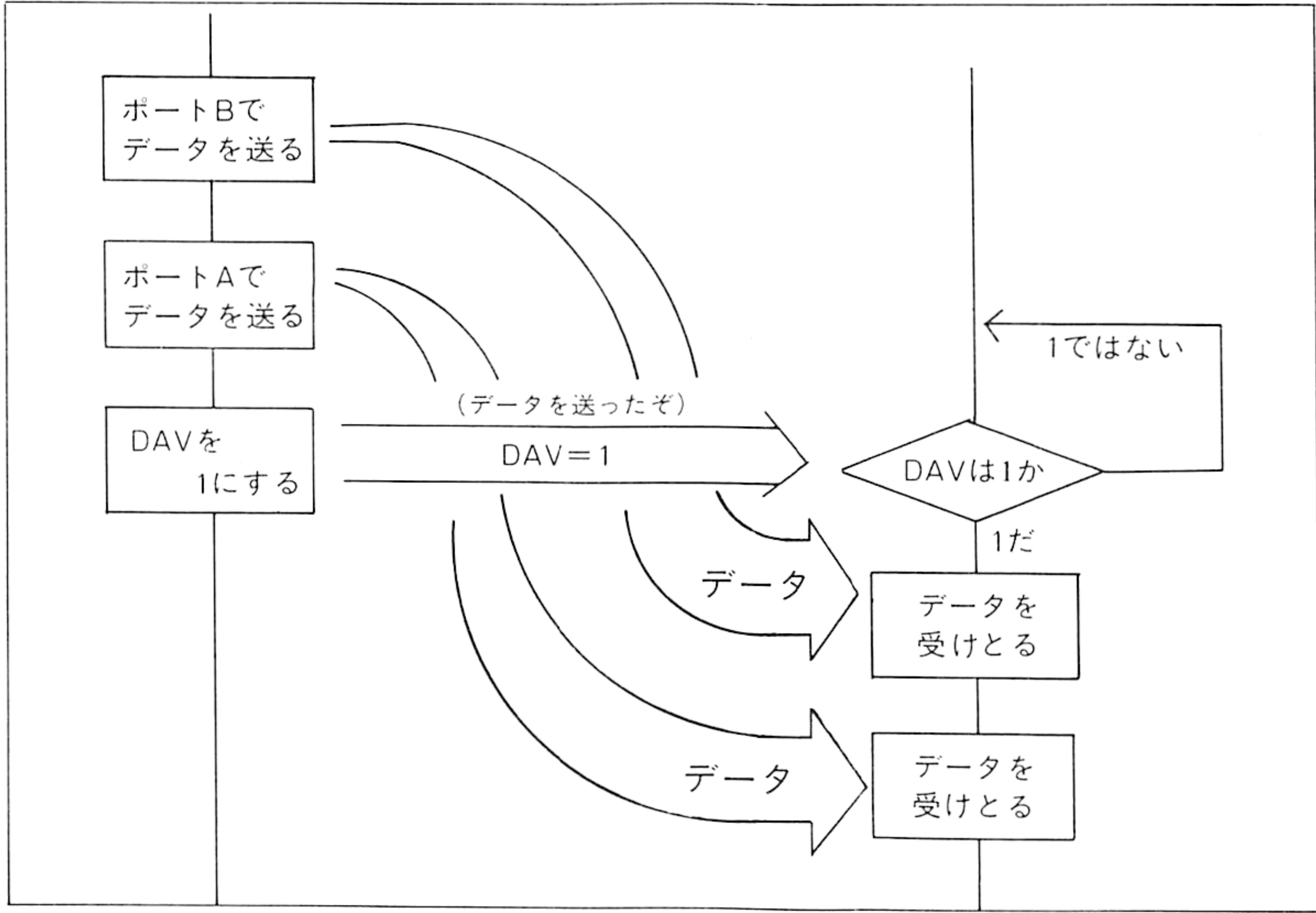
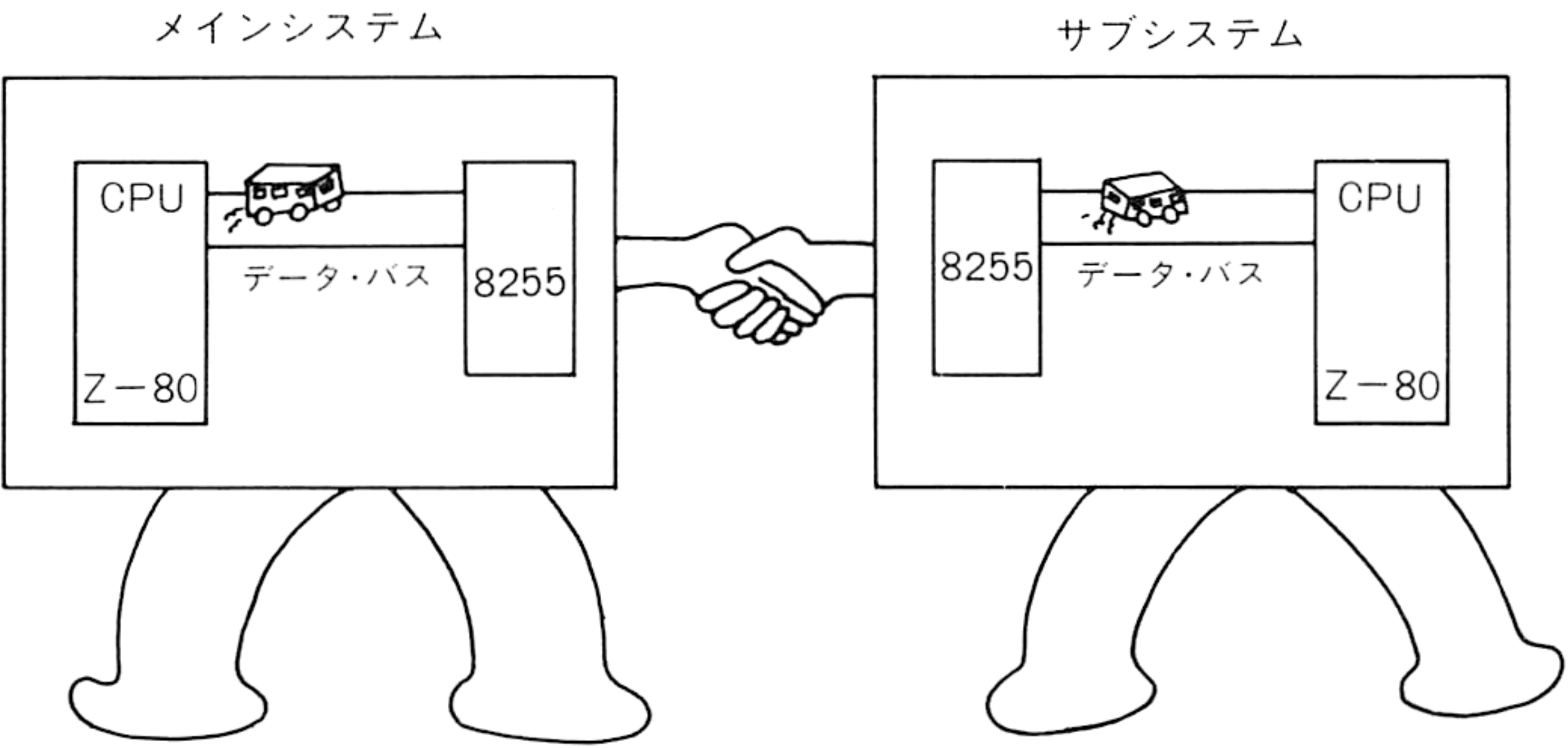
ポートBでデータを送る
ポートAでデータを送る
DAVを1にする
（データを送ったぞ）
DAV＝1
1ではない
DAVは1か
1だ
データ
データを受けとる
データ
データを受けとる

出力ポートが2つのとき（ポートAとポートB）

図10－２－G

10-2-3　8255の使い方とハンドシェイクルーチンのコード化

メインシシステムとサブシステムは、二つの8255(正確には汎用パラレルインタフェースLSI，μPD8255AC-5)を介してつながっています。ハンドシェイクでのデータ・信号のやり取りは、この8255を使って行なうわけです。ですから、ハンドシェイクのアルゴリズムから実際のプログラムを作るには、8255の使い方を理解しなければなりません。

メインシステム　サブシステム
CPU　Z−80　データ・バス　8255　8255　データ・バス　CPU　Z−80

8255には8ビットの入出力ポートが3個あり、それぞれをポートA，ポートB，ポートCと呼ぶことは前に述べました。各ポートは、その入出力モードを自由に設定でき、特にポートCは上位4ビットと下位4ビットを別々に設定できるようになっています。また、ポートCについてのみ、ビット単位でビットセット、リセット(1にしたり0にしたりすること)する機能があります。

二つの8255は、共にポートAをデータの入力、ポートBをデータの出力、ポートCの下位4ビットを信号の入力、上位4ビットを信号の出力に使うようにハード、ソフト共に設定されています。図10-2-Hを見て下さい。ハード的(回路的)にはこのようなつながり方をしています。メインシステム・サブシステムどちらから見ても、ポートBでデータを出力してポートAでデータを受け取るようになっているのです。

CPU　ポートA　ポートB　ポートC
PC_0(DAV)　PC_1(RFD)　PC_2(DAC)　PC_3　PC_4(DAV)　PC_5(RFD)　PC_6(DAC)　PC_7(ATN)
μPD8255　メインシステム

ポートA　ポートB　CPU　ポートC
PC_0(DAV)　PC_1(RFD)　PC_2(DAC)　PC_3(ATN)　PC_4(DAV)　PC_5(RFD)　PC_6(DAC)　PC_7
μPD8255　サブシステム

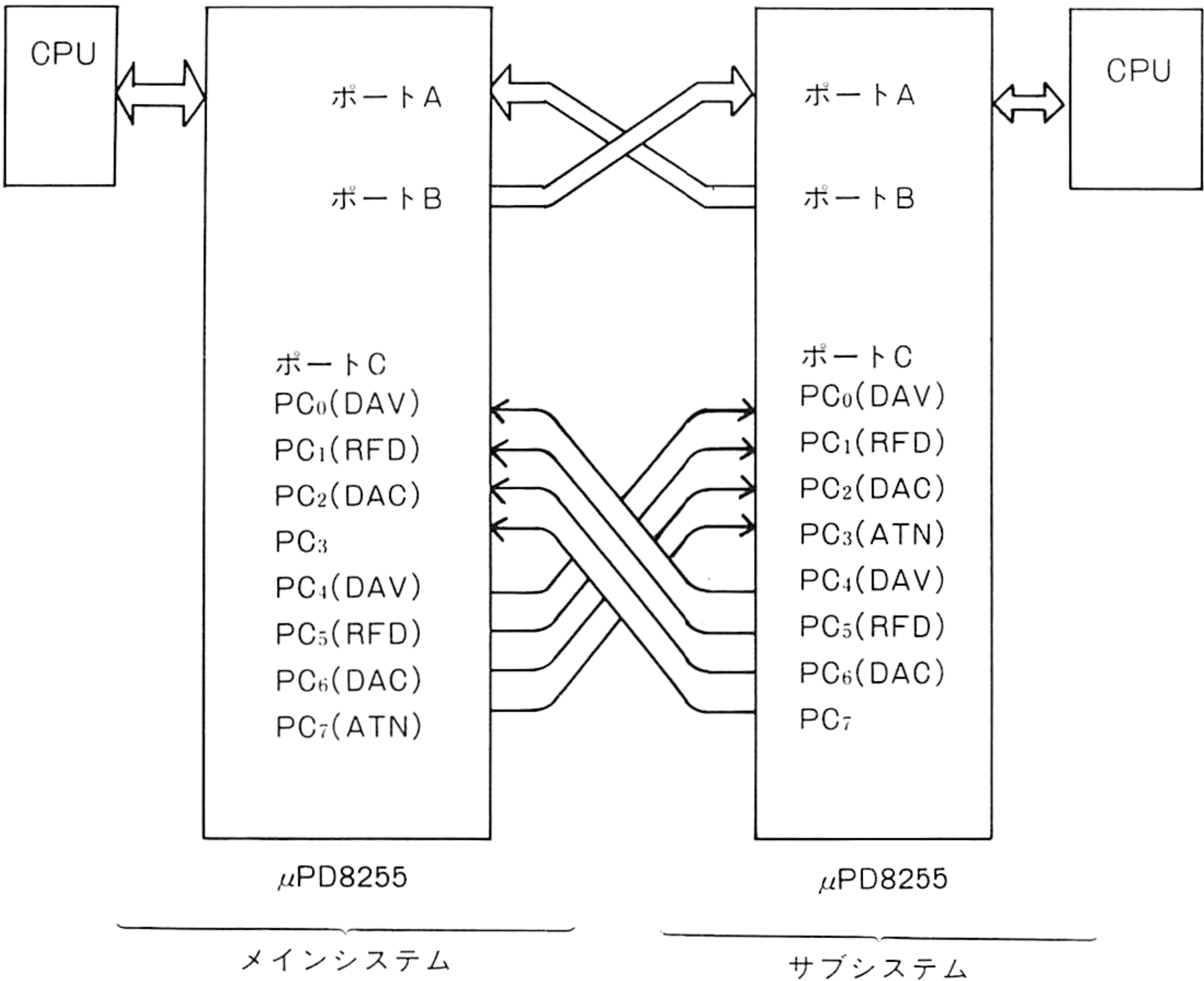



図10-2-H

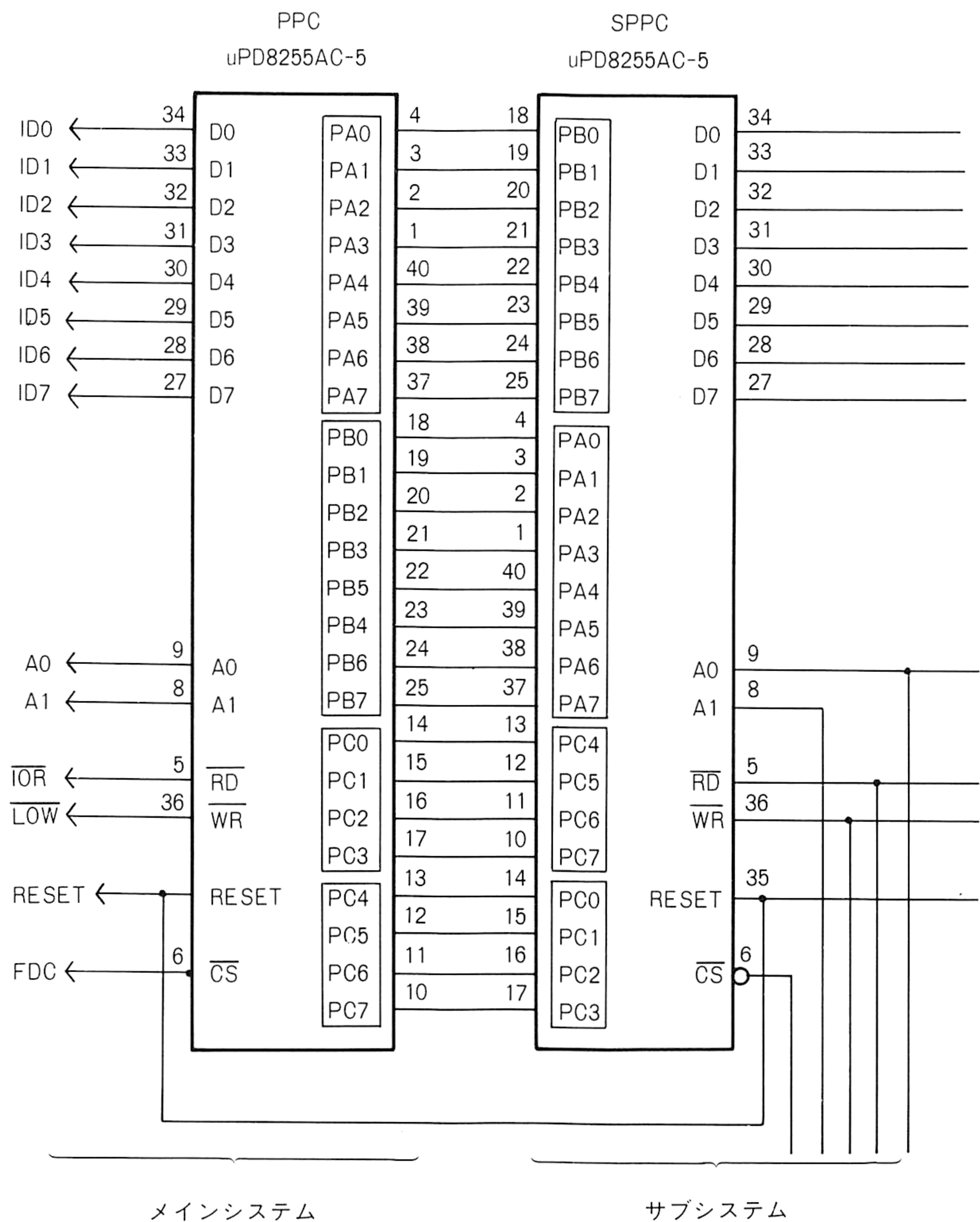



図10－２－１　8255によるインタフェース

それでは8255の初期設定を行なってみましょう。

8255のコントロールポートはI／OアドレスのFFHになっています。コントロールポートにコントロールワードを出力(OUT)すれば、初期設定は完了します。

コントロールワードの詳細は次のようになっています。

コントロールポート OFFH

コントロールワード = D7 D6 D5 D4 D3 D2 D1 D0

ビット	項目	値	内容	設定
グループB				
D0	ポートC(下位)	0	出力	1 ポートC(下位) 入力
		1	入力	
D1	ポートB	0	出力	0 ポートB 出力
		1	入力	
D2	モード	0	モード0	0 モード0
		1	モード1	
グループA				
D3	ポートC(上位)	0	出力	0 ポートC(上位) 出力
		1	入力	
D4	ポートA	0	出力	1 ポートA 入力
		1	入力	
D6 D5	モード	00	モード0	00 モード0
		01	モード1	
		1×	モード2	
D7	機能制御	0	ビットセット、リセット	1 モード選択
		1	モード選択	

10010001(2進) ＝91(16進)

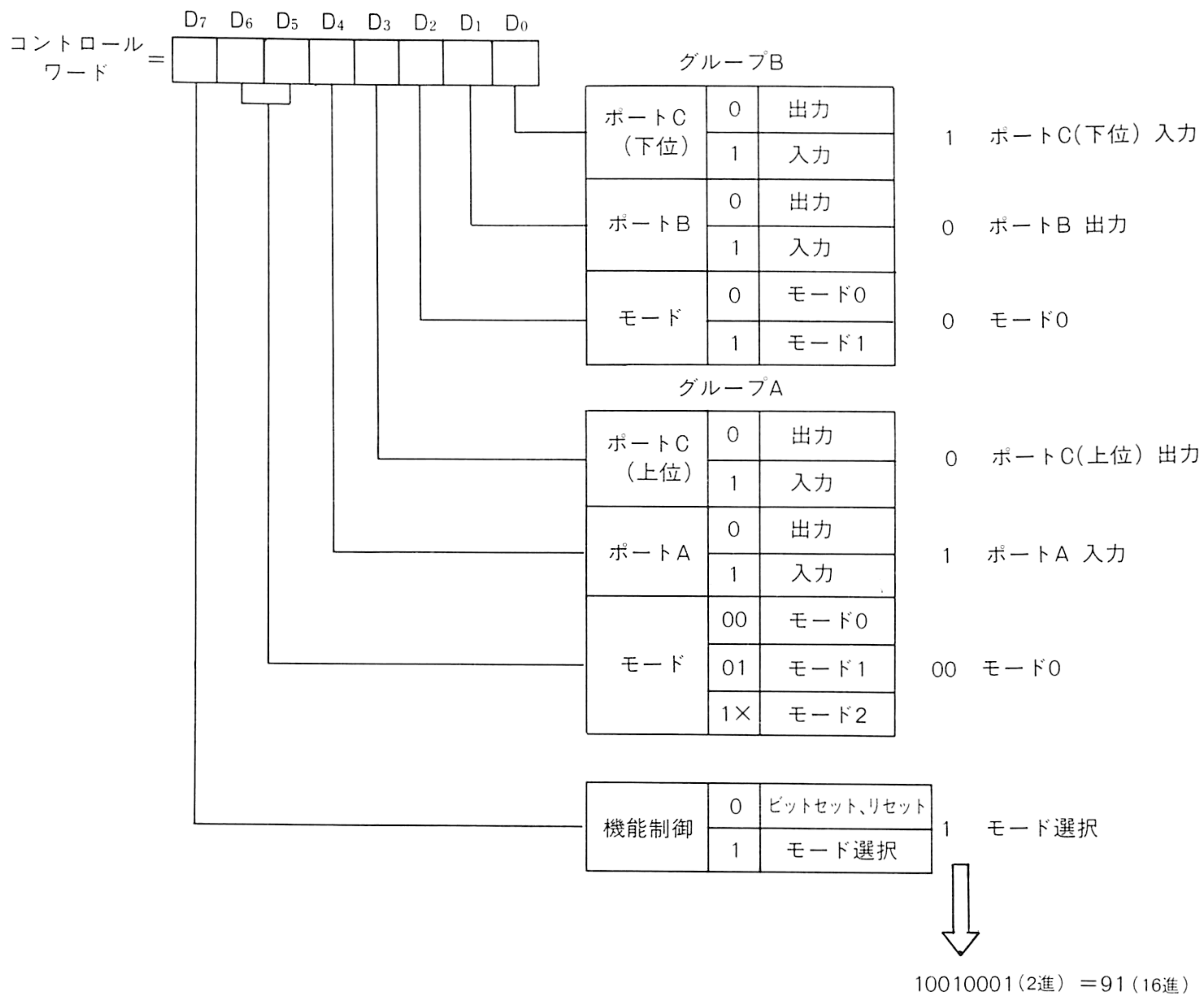



図10－２－J

8255はポートA及びポートCの上位４ビット、ポートB及びポートCの下位４ビットをグループA，Bと二つに分け、それぞれについて、いくつかのモードを選択できるようになっています。

モード０：基本的な入出力ポート

モード１：コントロール信号、ステータス信号による制御を伴う入出力ポート

モード２：双方向データを扱う入出力ポート

モード１，２はハード的にハンドシェイクを行なうモードで、8801mkⅡでは使っていません。グループA，B共にモード０を選択します。そして各ポートの入出力を決めると、コントロールワードは91Hになります。

OUT　&HFF，&H91

とすると8255の初期設定は完了します。この設定はメインシステム、サブシステム共にイニシャライズのときに行なわれます。

各ポートへのデータの入出力は、定められたI／OアドレスをIN，OUT(入出力命令)することで行なえます。I／Oアドレスはメイン、サブシステムとも同じ値で、ポートAはFCH、ポートBはFDH、ポートCはFEHになっています。例えばポートBからデータを出力すると

きはOUT &HFD, BDATA、ポートAからデータを入力するときは、ADATA＝INP(&HFC)を実行します。

ポートCはハンドシェイクの制御信号RFD, DAV, DACに使うので、入力はポートAの場合と同じですが、出力はビットセット／リセット機能を使います。

コントロールワードの最上位ビットが0のとき、コントロールワードはモード選択ではなく、ビット単位でのセット／リセットを表わします。

コントロールポート　0FFH

D7	D6	D5	D4	D3	D2	D1	D0
	×	×	×				

D0:

ビットセット リセット	0	リセット
	1	セット

D1〜D3:

ポートCのビット選択								
ビットNo.	0	1	2	3	4	5	6	7
D1	0	1	0	1	0	1	0	1
D2	0	0	1	1	0	0	1	1
D3	0	0	0	0	1	1	1	1

D7:

機能制御	0	ビットセット，リセット
	1	モード選択

……0ビットセット，リセット

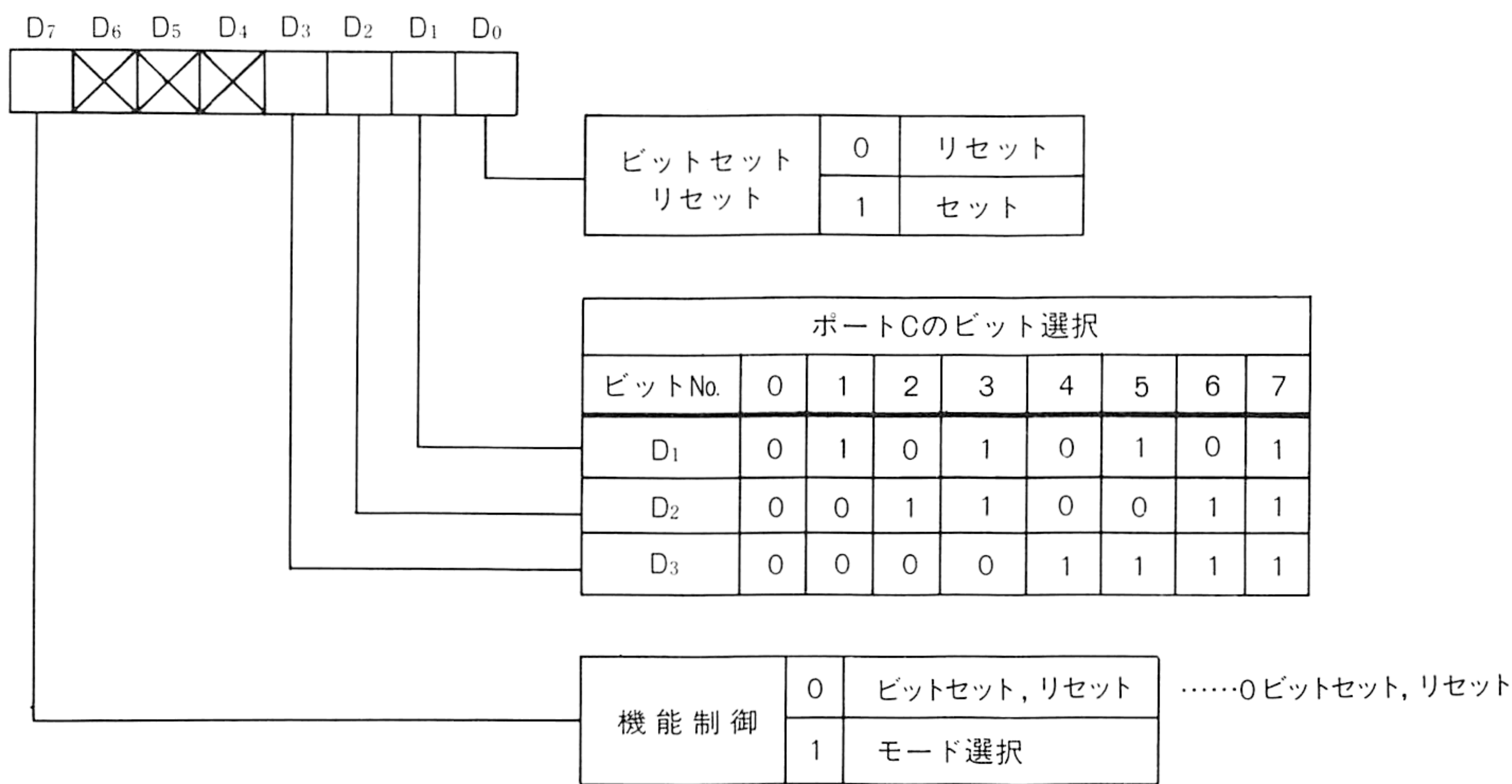



図10－2－K

ビット1～3で、ポートCの何ビット目かを指定し、ビット0でセット(1にする)するか、リセット(0にする)するかを決めます。

例えば、ポートCの第5ビットを1にしたければ、コントロールポートに00001011(2進数)＝11(10進数)を出力します。8801／mkⅡの場合、ポートCの第4ビットをDAV、第5ビットをRFD、第6ビットをDAC、第7ビットをATNの相手側への出力として用い、第0～第3ビットをDAV, RFD, DAC, ATNの入力として用います。

図10-2-Hを見て下さい。ハード的に2つの8255の間で、メインシステムのポートCの第4～7ビットはサブシステムのポートCの第0～3ビット、サブシステムのポートCの第4～7ビットはメインシステムの第0～3ビットに接続されています。そしてコントロールワードは91Hと同じなので、どちらから見ても第4～7ビットで出力した信号は、相手側の第0～3ビットへ送られるのです。

メインシステムでDAVを1にして、サブシステムでそれをチェックするマシン語のプログラムを考えてみましょう。

DAV信号への出力は第4ビットなので、コントロールポートへ00001001(2進数)＝9(10進数)をOUTすれば、DAVは1になります。それをサブシステム側でチェックするには、IN　A,(0FEH)(サブシステム・メインシステム共にポートCのI／Oアドレスは0FEH)を実行し、ポートCの入力データ(Accに入っている)の第0ビットが0か1かを判断します。もし第0ビッ

トが1なら、メインシステムよりDAV＝1の信号が来たことになります。

メインシステム側

```
CM  EQU  0FFH
    LD   A，9
    OUT  (CM)，A
```

サブシステム側

```
PORTC    EQU  0FEH
CKDAV1： IN   A，(PORTC)
         AND  00000001B   }
         JR   Z，CKDAV1   } (*)
```

DAV＝1

メイン　ポートC　ビット4

サ ブ　ポートC　ビット0

DAV

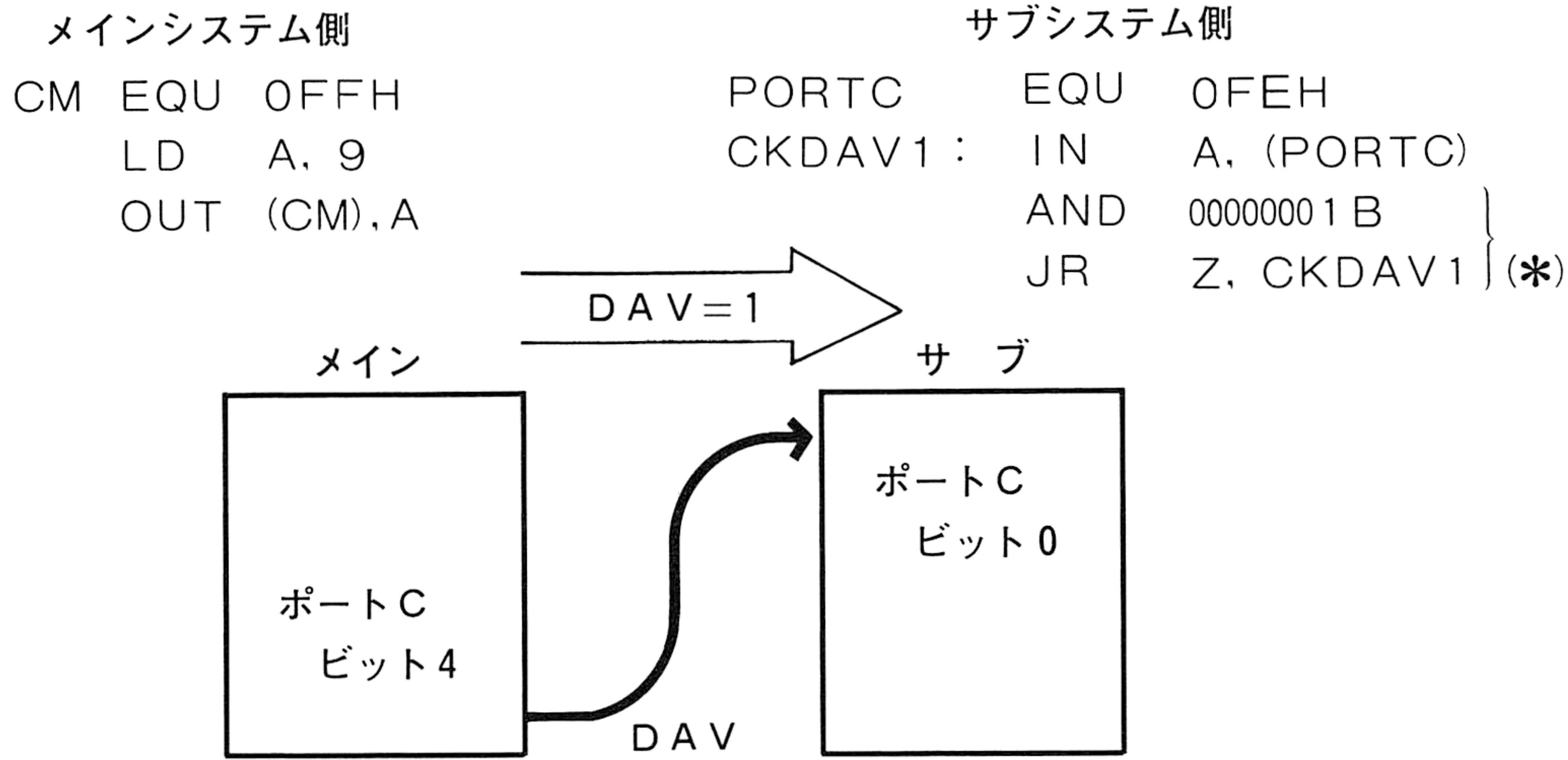



図10－2－L

＊DAVが0のときAND 1を実行すると結果は0となり、Zero flagが1になるので、CKDAV1にジャンプします。DAVが1のときはAND 1を実行すると結果は1となり、Zero flagが0になるので、ジャンプしません。

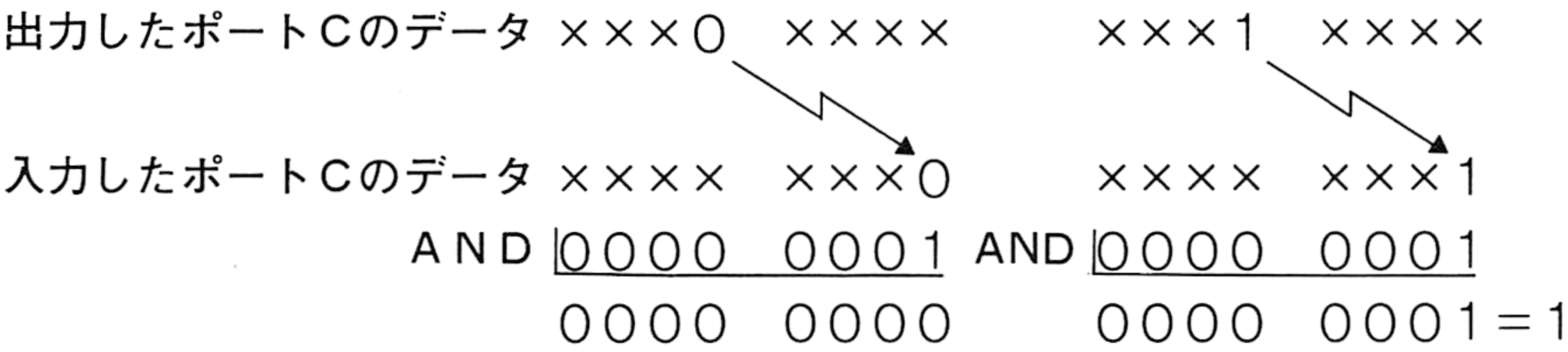
```
出力したポートCのデータ ×××0 ××××      ×××1 ××××
入力したポートCのデータ ×××× ×××0      ×××× ×××1
                   AND 0000 0001  AND 0000 0001
                       0000 0000      0000 0001＝1
```

ここまでくるとハンドシェイクのアルゴリズムをプログラムすることができます。

図10-2-BをBASICのプログラムで書いてみましょう。

データを送る側

```
100 CM=&HFF:PA=&HFC:PB=&HFD:PC=&HFE
110 IF(INP(PC) AND 2)=0 THEN 110
120 OUT PB，A
130 OUT CM，9

150 IF (INP(PC) AND 4)=0 THEN 150
160 IF (INP(PC) AND 2)=1 THEN 160
170 OUT CM，8
180 IF (INP(PC) AND 4)=1 THEN 180
```

データを受け取る側

```
100 CM=&HFF:PA=&HFC:PB=&HFD:PC=&HFE
110 OUT CM，11

130 IF(INP(PC) AND 1)=0 THEN 130
140   A=INP(PA)
150 OUT CM，13
160 OUT CM，10
170 IF (INP(PC) AND 1)=1 THEN 170
180 OUT CM，12
```

〔解説〕
100 ：コマンドポート　3つのポートの設定
110 ：RFD＝1
120 ：データを送る
130 ：DAV＝1
140 ：データを受け取る
150 ：DAC＝1
160 ：RFD＝0
170 ：DAV＝0
180 ：DAC＝0

サブシステムの制御はコマンドを送ることで行なわれます。サブシステムのROMに入っているプログラムは30種類のコマンドを持っていて、コマンド番号とそのコマンドの実行に必要なデータを送ることで、簡単にディスクを動かせるようになっているのです。

実際のハンドシェイクでは、このコマンドとデータを区別するためにATN(ATtentioN)と呼ばれる信号を用います。

メインシステムからサブシステムへコマンドを送るときのみハンドシェイクの先頭でATNを１にします。それ以外の場合、データを送るときとデータを受け取る場合はATNは０のままにしておきます。これ以外は図10-２-Cのアルゴリズムとまったく同じです。ATNはポートCの第7ビットで出力し、第3ビットで入力するようになっています。以下にサブシステムとハンドシェイクを行なうためのプログラムを載せておきます。

送り側
メイン・システム
1ではない
RFDは1か
1だ
データを送る
DAVを1にする
1ではない
DACは1か
1だ
DAVを0にする
0ではない
DACは0か
0だ
終 了

ATNを加える

送り側
メイン・システム
ATNを1にする
1ではない
RFDは1か
1だ
ATNを0にする
データを送る
DAVを1にする
1ではない
DACは1か
1だ
DAVを0にする
0ではない
DACは0か
0だ
終 了

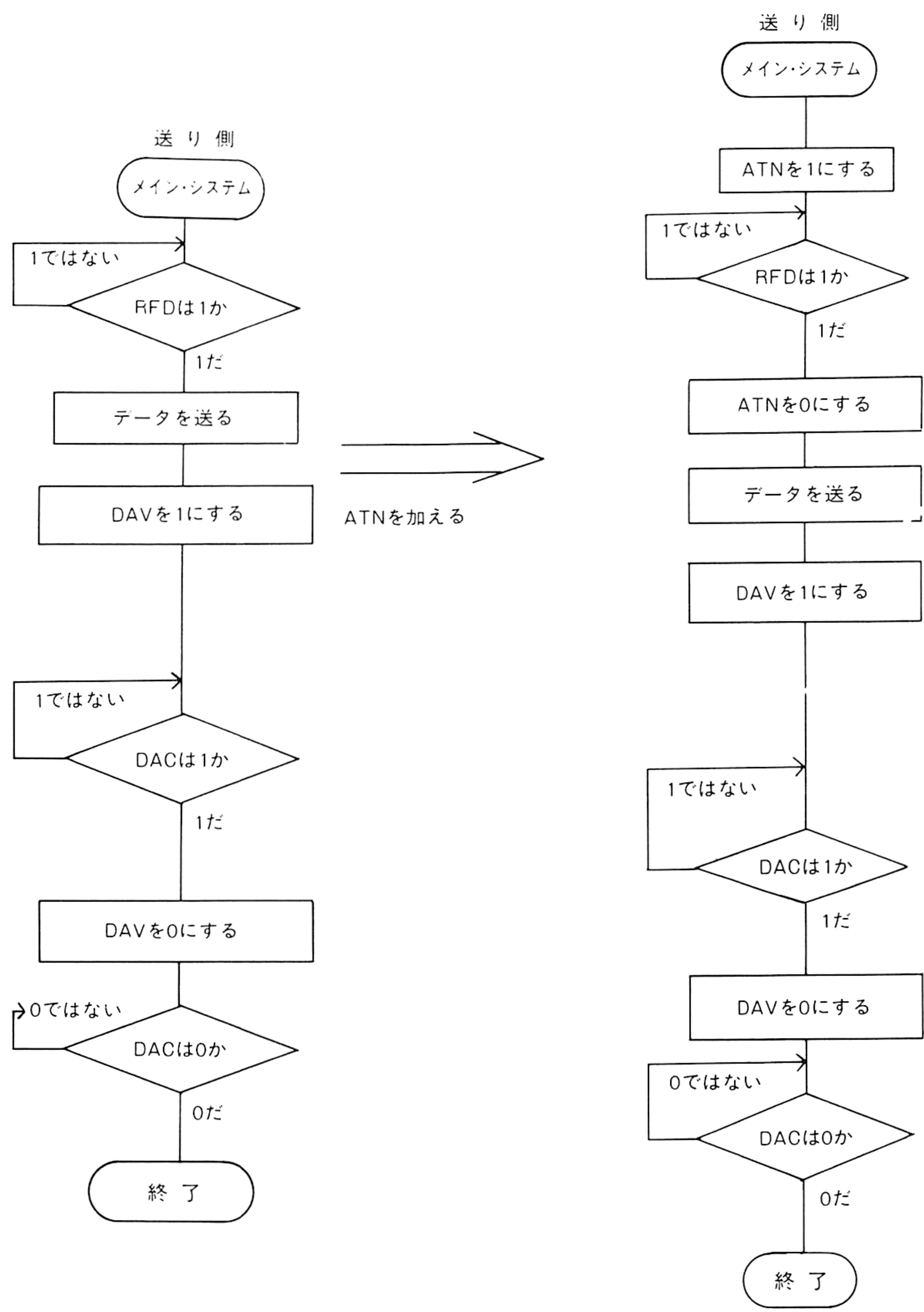



図10－２－M

サブシステムとのハンドシェイクルーチン

1）＊INTHS　：変数の初期化

```
1000 *INTHS
1010 CM=&HFF:PC=CM-1:PB=CM-2:PA=CM-3
1020 RETURN
```

2）＊SNDCOM　：コマンドを送る
　＊SNDPAR　：データを送る

```
1030 *SNDCOM
1040 OUT CM,&HF          →ATNを1にする
1050 *SNDPAR
1060 IF (INP(PC) AND 2)=0 THEN 1060
1070 OUT CM,&HE          →ATNを0にする
1080 OUT PB,A
1090 OUT CM,9
1100 IF (INP(PC) AND 4)=0 THEN 1100
1110 OUT CM,8
1120 IF (INP(PC) AND 4)=1 THEN 1120
1130 RETURN
```

3）＊REVPAR　：データを受け取る

```
1140 *REVPAR
1150 OUT CM,&HB
1160 IF (INP(PC) AND 1)=0 THEN 1160
1170 OUT CM,&HA
1180 A=INP(PA)
1190 OUT CM,&HD
1200 IF (INP(PC) AND 1)=1 THEN 1200
1210 OUT CM,&HC
1220 RETURN
```

1000行からのサブルーチンでコマンドポート、ポートA，B，CのI／Oアドレスの初期設定を行ないます。このルーチンは最初にコールします。1030行からのサブルーチンは、サブシステムへコマンドを送るハンドシェイクルーチンで、変数Aにコマンド番号を入れ、コールします。1050行からのサブルーチンはデータを送るルーチンです。そして1140行からのサブルーチンは、サブシステムよりデータを受け取るルーチンで、受け取ったデータは変数Aに入ります。

高速ハンドシェイクは、1バイトのコマンドだけを送ることは出来ません。必ずデータの転送のときのみ使われます。ですからATN信号は必要なく、図10-2-Eのアルゴリズムどおりです。

4）＊SNDPAR 2 ：高速ハンドシェイクでデータを送る

```
1230 *SNDPAR2
1240 IF (INP(PC) AND 2)=0 THEN 1240
1250 OUT PB, A1
1260 OUT CM, 9
1270 IF (INP(PC) AND 4)=0 THEN 1270
1280 OUT PB, A2
1290 OUT CM, 8
1300 IF (INP(PC) AND 4)=1 THEN 1300
1310 RETURN
```

5）＊REVPAR 2 ：高速ハンドシェイクでデータを受け取る

```
1320 *REVPAR2
1330 OUT CM, &HB
1340 IF (INP(PC) AND 1)=0 THEN 1340
1350 OUT CM, &HA
1360 A1=INP(PA)
1370 OUT CM, &HD
1380 IF (INP(PC) AND 1)=1 THEN 1380
1390 A2=INP(PA)
1400 OUT CM, &HC
1410 RETURN
```

1230行からのサブルーチンは、変数Ａ１，Ａ２のデータをサブシステムへ送るルーチンです。1320行からのサブルーチンは、データを受け取り、変数A1，A2に入れるプログラムです。

最後に、ハンドシェイクの高速化で述べた、データ量を2倍にする方法について考えてみましょう。データを送る側は、ポートＡもポートＢも、出力にセットします。つまり、コマンドポートへ81Hを出力します。そして、データを受け取る側は、ポートA，B共に入力にセットします。つまりコマンドポートへ93Hを出力します。これで送信側からポートA，Ｂへ出力されたデータは、受信側のポートＢ，Ａで入力出来るようになります。

この方法は送受信の方向が変わるたびにコマンドワードを設定しなおさなければなりませんが、簡単に転送量を2倍に出来ます。

データを送る側

メイン・システム

コントロールホートへ81Hを出力

RFDは1か
1ではない
1だ

ホートBにデータを出力
ホートAにデータを出力
DAVを1にする

DACは1か
1ではない
1だ

ホートBにデータを出力
ホートAにデータを出力
DAVを0にする

DACは0か
0ではない
0だ

RFDは0か
0ではない
0だ

コントロールホートへ91Hを出力

終　了

データをうけとる側

サブ・システム

コントロールホートへ93Hを出力

RFDを1にする

DAVは1か
1ではない
1だ

ホートAよりデータを入力
ホートBよりデータを入力
DACを1にする

DAVは0か
0ではない
0だ

ホートAよりデータを入力
ホートBよりデータを入力
DACを0にする

RFDを0にする

コントロールホートへ91Hを出力

終　了

RFV＝1
DAV＝1
データ
データ
DAC＝1
DAV＝0
データ
データ
DAC＝0
RFD＝0

上の枠内のくり返し

図10－2－N

10-3 サブシステムのコントロールコマンド

この節では、サブシステムのコントロールコマンドについて機能別に説明していきます。

その前に、中で使われるコマンドフォーマットの表の見方を説明しておきます。これは、各コマンドに必要なパラメータをどのような順番で入出力するかを表にしたもので、以下の規則に従い、左から右へ実行していきます。

....コマンドか、1バイトのデータをサブシステムへ送ります。

....2バイトのデータをサブシステムへ送ります。

....必要な量のデータをサブシステムへ送ります。

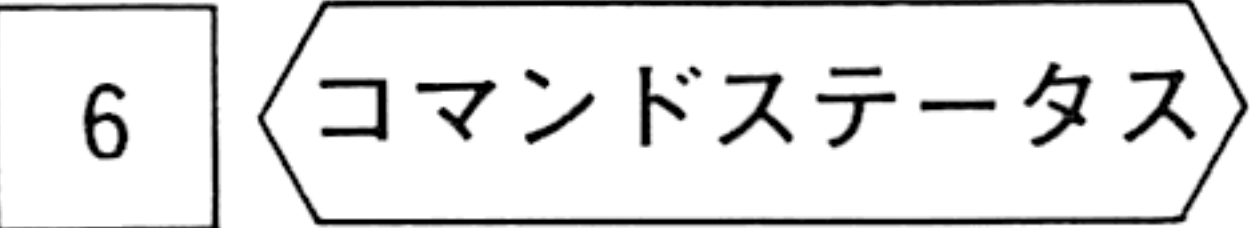
....必要な量のデータを高速ハンドシェイクで送ります。

....1バイトのデータを受け取ります。

....2バイトのデータを受け取ります。

....必要な量のデータを受け取ります。

....必要な量のデータを高速ハンドシェイクで受け取ります。

例

6 コマンドステータス

コマンド番号6をサブシステムへ送った後、1バイト(コマンドステータス)受け取ります。

次に用語の問題について第9章に補足しておきます。

この章では、ディスクドライブの同義語としてドライブ、ユニット(ドライブユニット)を使っています。これらは、すべて同じ物を示すのですが、ドライブは1，2，3，4と指定するのに対し、ユニットは0，1，2，3と指定します。サブシステムの内部では、ドライブ番号を0から数えるため、このような区別をつけました。

トラック番号は後者の考え方を使い、またシリンダという言葉も使います。

最後に各コマンドのサンプルプログラムの共通サブルーチンプログラムと、コマンド番号順の索引を載せておきます。

リスト　10－1　共通サブルーチン

```
1000 *INTHS
1010 CM=&HFF:PC=CM-1:PB=CM-2:PA=CM-3
1020 RETURN
1030 *SNDCOM
1040 OUT CM,&HF
1050 *SNDPAR
1060 IF (INP(PC) AND 2)=0 THEN 1060
1070 OUT CM,&HE
1080 OUT PB,A
1090 OUT CM,9
1100 IF (INP(PC) AND 4)=0 THEN 1100
1110 OUT CM,8
1120 IF (INP(PC) AND 4)=1 THEN 1120
1130 RETURN
1140 *REVPAR
1150 OUT CM,&HB
1160 IF (INP(PC) AND 1)=0 THEN 1160
1170 OUT CM,&HA
1180 A=INP(PA)
1190 OUT CM,&HD
1200 IF (INP(PC) AND 1)=1 THEN 1200
1210 OUT CM,&HC
1220 RETURN
```

コマンド索引

10-3-1　サブシステムのメモリの内容を読む(Receive data)

サブシステムのメモリの内容をメインシステムへ送るコマンドは5つあります。

1)コマンド3と18

リードバッファにあるデータをメインシステムへ送るコマンドです。 コマンド3は、 コマンド2でリードバッファに読み込まれたデータを転送するためのもので、単独で使われることはありません。コマンド18は、データの転送に高速ハンドシェイクを用いる以外は、コマンド3と同じです。(10-3-3参照)

コマンドフォーマット

3　データ
コマンドNo　データを受け取る

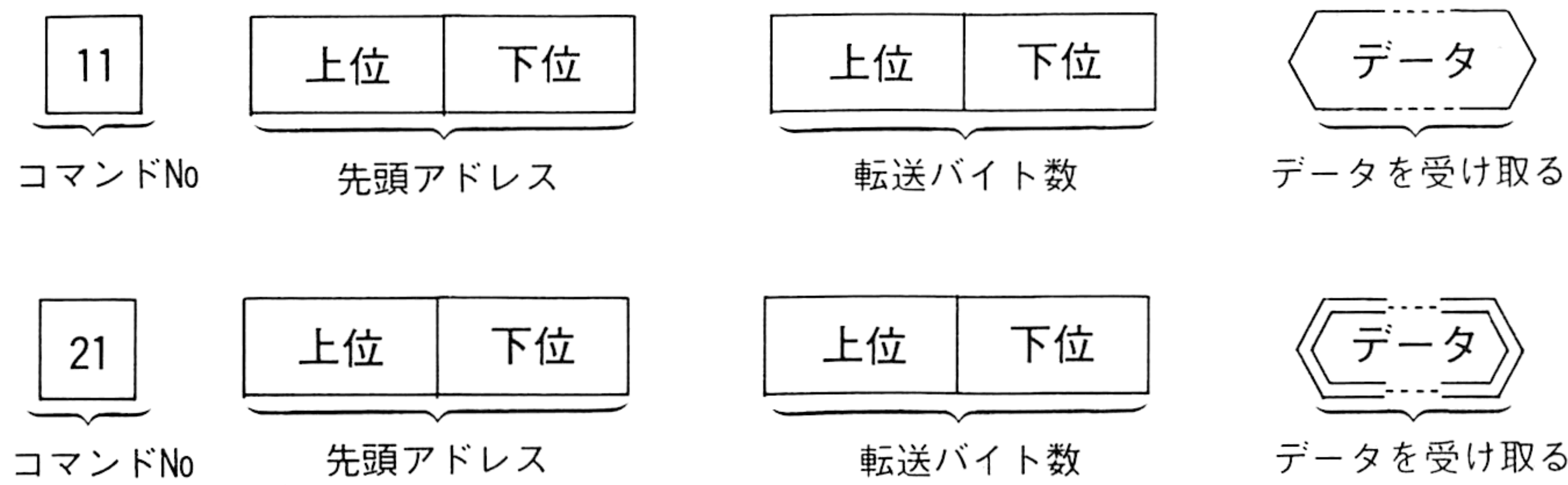
18　データ
コマンドNo　データを受け取る

2)コマンド11と21

先頭アドレスと転送バイト数を指定することで、任意のアドレスのデータをメインシステムへ送ることができます。(コマンド21は、コマンド11の高速ハンドシェイク版です。)

コマンドフォーマット

11	上位	下位	上位	下位	データ
コマンドNo	先頭アドレス		転送バイト数		データを受け取る

21	上位	下位	上位	下位	データ
コマンドNo	先頭アドレス		転送バイト数		データを受け取る

コマンド11を使って、サブシステムの任意のアドレスから256バイトをメインシステムのC000H番地に転送してみましょう。

リスト10-2

```
100 CLEAR ,&HBFFF
110 GOSUB *INTHS
120 INPUT "Input first address ";DUM$:FADD$=RIGHT$("0000"+DUM$,4)
130 A=11                         :GOSUB *SNDCOM  コマンド番号
140 A=VAL("&H"+LEFT$(FADD$,2))   :GOSUB *SNDPAR  先頭アドレスの上位
150 A=VAL("&H"+RIGHT$(FADD$,2))  :GOSUB *SNDPAR       〃      下位
160 A=1                          :GOSUB *SNDPAR  転送バイト数の上位
170 A=0                          :GOSUB *SNDPAR       〃      下位
180 FOR I=&HC000 TO &HC0FF
190   GOSUB *REVPAR:POKE I,A                     サブシステムから送られる
200 NEXT                                         データを受け取る
210 PRINT "Complete"
220 END
```

※1000～1220:共通サブルーチン(182P)

このプログラムを実行して、サブシステムの0番地～255番地の内容を転送してみると、次のようになります。

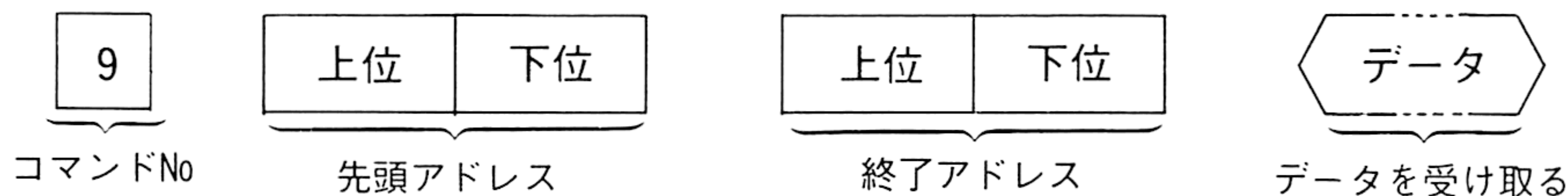
リスト10－3 コマンド11実行例

```
run
Input first address ? 0
Complete
Ok
mon

h) dc000, c07f
C000 C3 7E 00 00 00 00 00 00 F3 CD 22 7F C3 4E 00 00
C010 C3 D9 06 00 00 00 00 00 C3 0F 07 00 69 00 00 00
C020 C3 A7 02 00 C3 D9 06 C3 0F 07 C3 C1 00 C3 D5 02
C030 C3 D6 02 C3 96 03 C3 96 01 C3 E7 01 C3 C2 C2 C3
C040 9A 02 C3 A7 02 C3 63 02 C3 41 07 C3 7B 07 E3 2B
C050 22 2A 7F E1 ED 73 28 7F 31 42 7F F5 C5 D5 E5 08
C060 D9 F5 C5 D5 E5 DD E5 FD E5 ED 57 32 2C 7F 31 00
C070 80 CD 17 06 2A 2A 7F 7C DF 7D DF C3 C1 00 31 00
h)
```

3)コマンド9

先頭アドレスと終了アドレスに1を加えた値を指定することで、先頭アドレスから終了アドレスまでのメモリーの内容を転送します。

コマンドフォーマット

9	上位	下位	上位	下位	データ
コマンドNo	先頭アドレス		終了アドレス		データを受け取る

コマンド9を使ってサブシステムのメモリーダンプを取ってみて下さい。

リスト10－4

```
100 GOSUB *INTHS
110 INPUT "Input first address ";DUM$:FADD=VAL("&H"+DUM$)
120 INPUT "Input end   address ";DUM$:EADD=VAL("&H"+DUM$)
130 FADD$=RIGHT$("0000"+HEX$(FADD),4)
140 EAD1$=RIGHT$("0000"+HEX$(EADD+1),4)
150 IF FADD<0 THEN FADD=FADD+65536!
160 IF EADD<0 THEN EADD=EADD+65536!
170 IF FADD>EADD THEN PRINT:GOTO 110
180 A=9                          :GOSUB *SNDCOM  コマンド9
190 A=VAL("&H"+LEFT$(FADD$,2))   :GOSUB *SNDPAR  先頭アドレスの上位
200 A=VAL("&H"+RIGHT$(FADD$,2))  :GOSUB *SNDPAR      〃      下位
210 A=VAL("&H"+LEFT$(EAD1$,2))   :GOSUB *SNDPAR  終了アドレス＋1の上位
220 A=VAL("&H"+RIGHT$(EAD1$,2))  :GOSUB *SNDPAR        〃        下位
230 COUNT=0
240 FOR I=FADD TO EADD
250   GOSUB *REVPAR                                    } データを受け取り
260   IF (COUNT MOD 16)=0 THEN GOSUB *DSPADD:GOTO 280  }    表示する
270 PRINT ",";RIGHT$("0"+HEX$(A),2);
280 COUNT=COUNT+1
290 NEXT
300 PRINT
310 END
400 *DSPADD
410 PRINT:PRINT RIGHT$("000"+HEX$(I),4);" : ";RIGHT$("0"+HEX$(A),2);
420 COUNT=0
430 RETURN
```

※1000～1220：共通サブルーチン(182P)

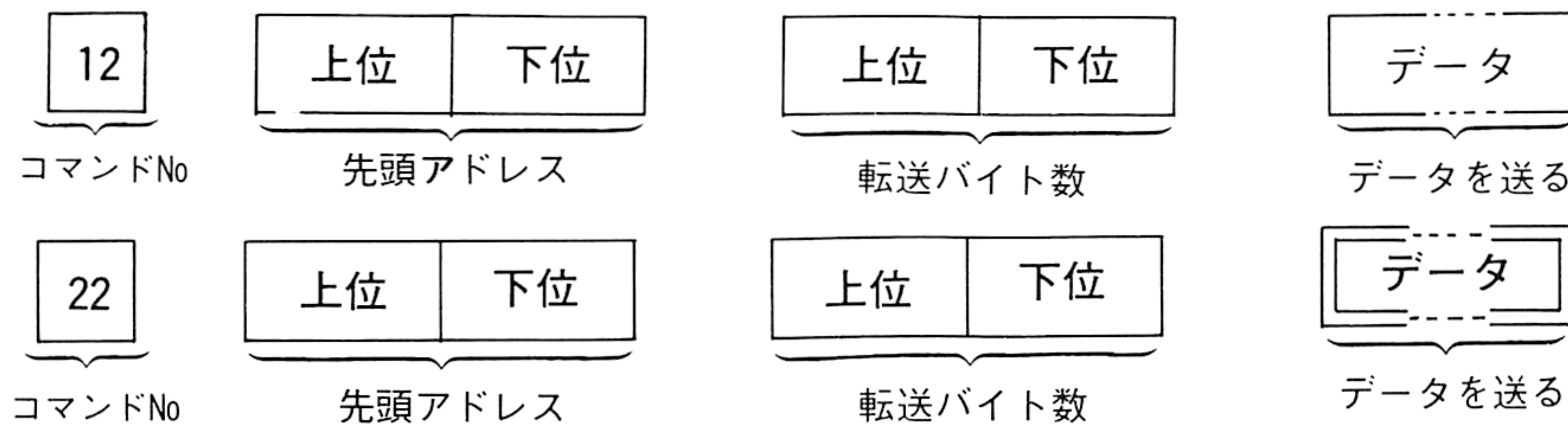
10-3-2　サブシステムへデータを送る(Send data)

コマンド12と22

サブシステムの任意のアドレスへデータを送るコマンドは、12と22の2つあります。それらは、先頭のアドレスと転送バイト数を指定することで、サブシステムへデータを送ることができます。(コマンド22はコマンド12の高速ハンドシェイク版です。)

コマンドフォーマット

12	上位 / 下位	上位 / 下位	データ
コマンドNo	先頭アドレス	転送バイト数	データを送る

22	上位 / 下位	上位 / 下位	データ
コマンドNo	先頭アドレス	転送バイト数	データを送る

コマンド12を使って、サブシステムのメモリーの内容を書き変えてみましょう。

リスト10-5

```
100 GOSUB *INTHS
110 INPUT "Input address ";DUM$:FADD$=RIGHT$("0000"+DUM$,4)
120 A=11                               :GOSUB *SNDCOM )
130 A=VAL("&H"+LEFT$(FADD$,2))         :GOSUB *SNDPAR | コマンド11で現
140 A=VAL("&H"+RIGHT$(FADD$,2))        :GOSUB *SNDPAR | 在のメモリの内
150 A=0                                :GOSUB *SNDPAR | 容を調べる
160 A=1                                :GOSUB *SNDPAR |
170                                    :GOSUB *REVPAR )
180 PRINT "Now,value is ";RIGHT$("00"+HEX$(A),2);"H"
190 INPUT "Input new value ";DUM$:CDATA=VAL("&H"+DUM$)
200 A=12                               :GOSUB *SNDCOM | コマンド12
210 A=VAL("&H"+LEFT$(FADD$,2))         :GOSUB *SNDPAR | 先頭アドレスの上位
220 A=VAL("&H"+RIGHT$(FADD$,2))        :GOSUB *SNDPAR |     〃      下位
230 A=0                                :GOSUB *SNDPAR | 転送バイト数の上位
240 A=1                                :GOSUB *SNDPAR |     〃      下位
250 A=CDATA                            :GOSUB *SNDPAR | データを送る
260 PRINT
270 GOTO 110
```

※1000～1220:共通サブルーチン(182P)

コマンド12実行例

```
run
Input address ? 6000
Now,value is FFH
Input new value ? 31
```

10-3-3　ディスクからデータを読み込む(Read data)

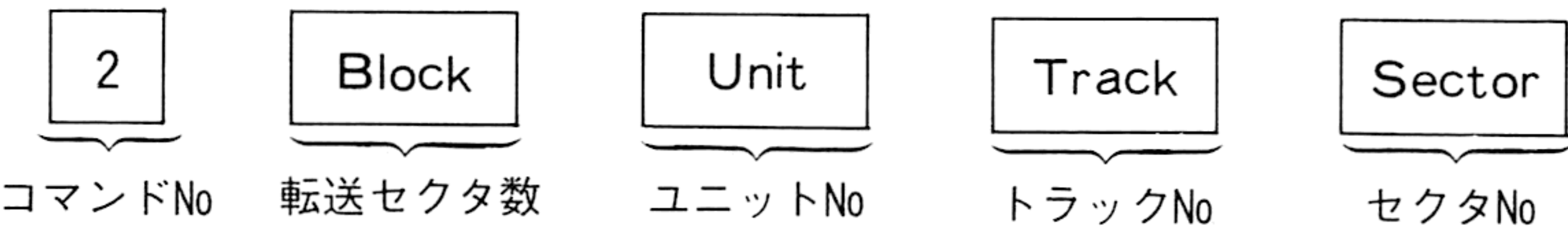
ディスクからデータを読み込み、サブシステムのメモリ上へ置くコマンドは2つあります。

1)コマンド2

ディスクからデータを読み込み、リードバッファに置きます。

転送セクタ数、ユニット番号(ドライブ番号−1)、トラック番号、セクタ番号を指定します。ユニット番号は、どのドライブユニットに入ったディスクをアクセスするか指定するもので、ドライブ1なら0、ドライブ2なら1を指定します。転送セクタ数は、読み込むセクタの数で、2つのトラックにまたがるような指定はできません。例えば、ユニット番号0(ドライブ1)のトラック0セクタ8から16セクタ(転送セクタ数)読み込むといったことはできません。5インチディスクの場合、セクタ番号の最大値は16なので、転送セクタ数は9以下になります。

コマンドフォーマット

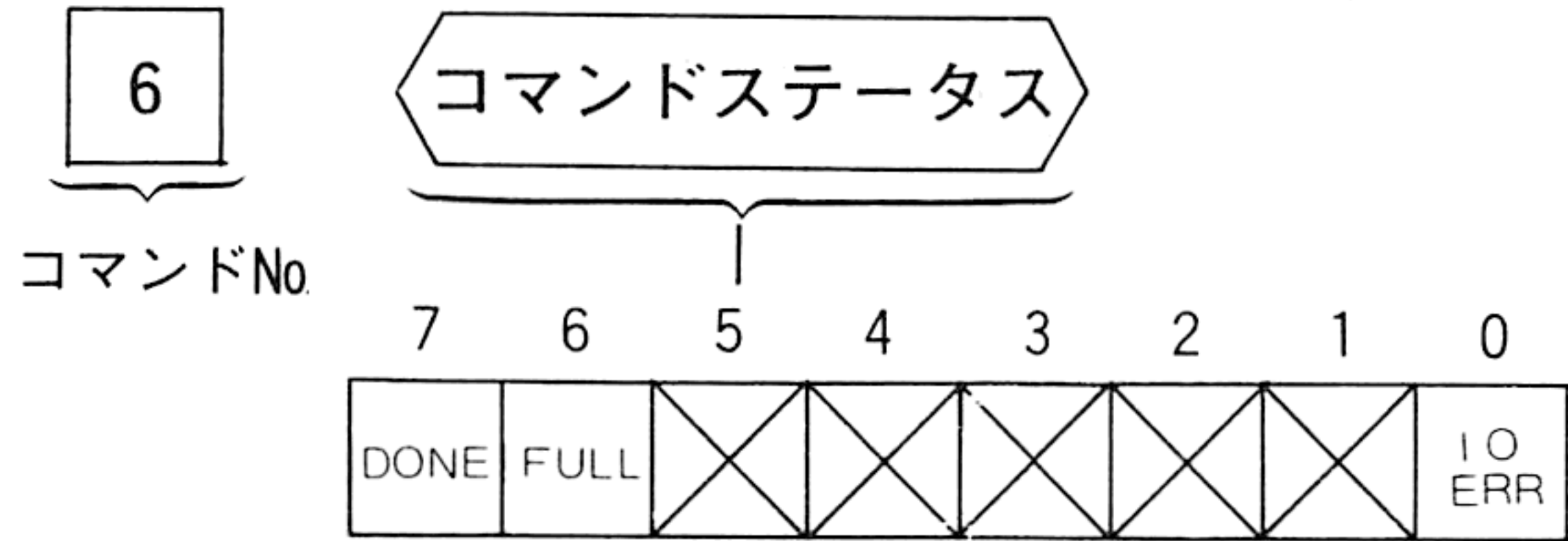
2	Block	Unit	Track	Sector
コマンドNo	転送セクタ数	ユニットNo	トラックNo	セクタNo

(転送セクタ数)＋(セクタ番号)≦17

読み込まれたデータは、5000H〜5FFFHのリードバッファに置かれ、転送セクタ数はワークエリア7F08Hに記憶されます。また、正常に読めた場合は、コマンドステータスのFULLフラグを1，IOERRフラグを0にします。正常に読めなかったときは、FULLフラグを0，IOERRフラグを1にします。コマンドステータスのFULLフラグが1で、IOERRフラグが0のとき、コマンド3を実行すると、7F08Hの転送セクタ数×256バイトのデータが送られてきます。

コマンドステータスは、コマンド6を使えば調べることができます。コマンド番号6をサブシステムへ送ると、1バイトのデータ(コマンドステータス)が送られて来ます。

コマンドフォーマット

6	コマンドステータス
コマンドNo	

7	6	5	4	3	2	1	0
DONE	FULL	×	×	×	×	×	IO ERR

IOERR.....処理中にI／Oエラーが起きたら1
　　　　　そうでなければ0
FULL......データ・アベイラブル・フラグ
　　　　　リード・バッファにデータがあるとき1、そうでないとき0
　　　　　例)コマンド2が正常に終了すると1になり、
　　　　　コマンド3でデータを送ってしまうと0になる。
DONE......ラストI／Oコンプリート・フラグ
　　　　　DISKがインテリジェント方式なので常に1

コマンド2，6，3を使ってディスクのデータをメインシステムのC000H以降にもってくるプログラムを載せておきます。

リスト10－6

```
100 GOSUB *INTHS
110 INPUT "Input drive  number (1,2)  ";DRIVE
120 IF DRIVE<>1 AND DRIVE<>2 THEN 110
130 INPUT "Input track  number (0-79) ";TRACK
140 IF TRACK<0 OR TRACK>79 THEN 130
150 INPUT "Input sector number (1-16) ";SECTOR
160 IF SECTOR<1 OR SECTOR>16 THEN 150
170 INPUT "Input block  number (1-16) ";BLOCK
180 IF BLOCK<1 OR BLOCK>16 OR SECTOR+BLOCK>17 THEN 170
190 BUFF=&HC000
200 '
210 A=2       :GOSUB *SNDCOM } コマンド2
220 A=BLOCK   :GOSUB *SNDPAR } 転送セクタ数を指定
230 A=DRIVE-1:GOSUB *SNDPAR } ドライブユニット番号
240 A=TRACK   :GOSUB *SNDPAR } トラック番号
250 A=SECTOR :GOSUB *SNDPAR } セクタ番号
300 '
310 A=6       :GOSUB *SNDCOM } コマンド6でエラーのチェックを行う
320            GOSUB *REVPAR }
330 IF (A AND 1)=1 THEN PRINT "Disk I/O error":BEEP:END
340 IF (A AND &H40)=0 THEN PRINT "Not exit data in read buff":BEEP:END
400 '
410 A=3       :GOSUB *SNDCOM          }
420 FOR I=0 TO BLOCK*256-1            } コマンド3でデータを受け取る
430   GOSUB *REVPAR:POKE I+BUFF,A     }
440 NEXT
450 PRINT "Complete"
460 END
```

※1000～1220：共通サブルーチン(182P)

コマンド2実行例

```
run
Input drive  number (1,2)  ? 1 } ドライブ1、トラック0の
Input track  number (0-79) ? 0 } セクタ1から1セクタを
Input sector number (1-16) ? 1 } C000H～C0FFHに読ん
Input block  number (1-16) ? 1 } でくる。
Complete
Ok
mon

h) dc000,c07f
C000 BE C0 1F 8E D6 BC 00 02 B8 5F 0E CD 1B 73 01 F4
C010 BB 00 98 B1 00 B6 00 B2 03 BE 00 10 8E C6 BD 00
C020 00 E8 24 00 A0 00 05 0C 50 A2 00 05 BE 60 00 8E
C030 DE C6 06 05 05 01 B8 00 E8 50 B8 02 00 50 CB B2
C040 01 80 F6 01 75 02 FE C1 8B FB BE 00 10 52 80 FA
C050 01 74 08 FE CA 81 EE 00 01 EB F3 5A 3B DE 72 02
C060 8B DE B8 50 56 CD 1B 72 97 87 DF 03 EF 2B DF 77
C070 CE C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

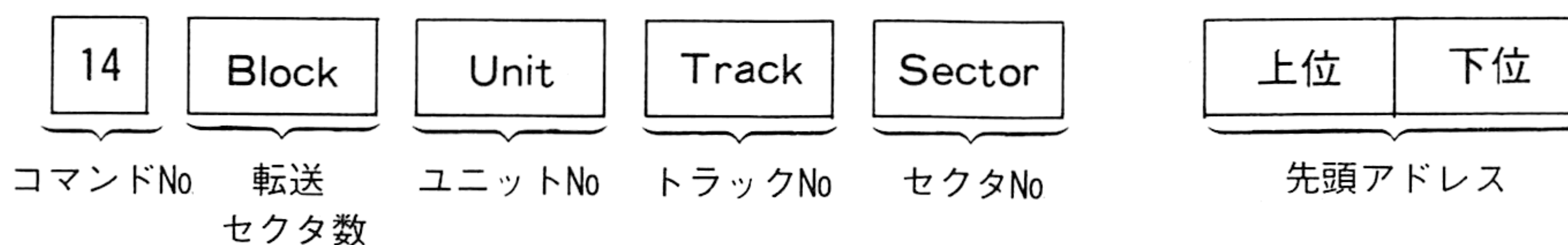
2)コマンド14

コマンド2は、読み込んだデータを5000H～5FFFH(リードバッファ)に置いていましたが、このコマンドでは、その先頭アドレスを自由に設定できます。ただし、ROMエリアや、ワークエリア(7F00H～7FFFH)に置くことはできません。これらのアドレスを使うと、ハングアップする可能性もあるので、注意して下さい。

他のパラメータはコマンド2と同じです。

正常に読めなかったときは、コマンドステータスのIOERRフラグが1になります。

コマンドフォーマット

14	Block	Unit	Track	Sector	上位	下位
コマンドNo	転送セクタ数	ユニットNo	トラックNo	セクタNo	先頭アドレス	

(転送セクタ数)＋(セクタ番号) ≦17

コマンド14, 6, 11を使い、ディスクのデータをメインシステムのC000H以降にもってくるプログラムを載せておきます。バッファの先頭アドレスは6000Hになっています。

リスト10－7

```
100 GOSUB *INTHS
110 INPUT "Input drive  number (1,2)  ";DRIVE
120 IF DRIVE<>1 AND DRIVE<>2 THEN 110
130 INPUT "Input track  number (0-79) ";TRACK
140 IF TRACK<0 OR TRACK>79 THEN 130
150 INPUT "Input sector number (1-16) ";SECTOR
160 IF SECTOR<1 OR SECTOR>16 THEN 150
170 INPUT "Input block  number (1-16) ";BLOCK
180 IF BLOCK<1 OR BLOCK>16 OR SECTOR+BLOCK>17 THEN 170
190 MAIN.BUFF=&HC000
200 SUB.BUFF=&H6000:SUB.BUFF$=RIGHT$("0000"+HEX$(SUB.BUFF),4)
210 '
220 A=14                           :GOSUB *SNDCOM  コマンド14
230 A=BLOCK                        :GOSUB *SNDPAR  転送セクタ数指定
240 A=DRIVE-1                      :GOSUB *SNDPAR  ドライブユニット番号
250 A=TRACK                        :GOSUB *SNDPAR  トラック番号
260 A=SECTOR                       :GOSUB *SNDPAR  セクタ番号
270 A=VAL("&H"+LEFT$(SUB.BUFF$,2)) :GOSUB *SNDPAR  バッファ先頭アドレス上位
280 A=VAL("&H"+RIGHT$(SUB.BUFF$,2)):GOSUB *SNDPAR       〃       下位
300 '
310 A=6                            :GOSUB *SNDCOM  } コマンド6で
320                                 GOSUB *REVPAR  } エラーのチェック
330 IF (A AND 1)=1 THEN PRINT "Disk I/O error":BEEP:END
400 '
410 A=11                           :GOSUB *SNDCOM
420 A=VAL("&H"+LEFT$(SUB.BUFF$,2)) :GOSUB *SNDPAR
430 A=VAL("&H"+RIGHT$(SUB.BUFF$,2)):GOSUB *SNDPAR
440 SIZE$=RIGHT$("0000"+HEX$(256*BLOCK),4)          コマンド11でバッファに
450 A=VAL("&H"+LEFT$(SIZE$,2))     :GOSUB *SNDPAR   あるデータを読む
460 A=VAL("&H"+RIGHT$(SIZE$,2))    :GOSUB *SNDPAR
470 FOR I=0 TO 256*BLOCK-1
480   GOSUB *REVPAR:POKE MAIN.BUFF+I,A
490 NEXT
500 PRINT "Complete"
510 END
```

※1000～1220：共通サブルーチン(182P)

コマンド14実行例

```
run
Input drive  number (1,2)  ? 1  ドライブ1、トラック0の
Input track  number (0-79) ? 0  セクタ1から1セクタを
Input sector number (1-16) ? 2  C000H～C0FFHに
Input block  number (1-16) ? 1  読んでくる。
Complete

mon
h〕dc000,c07f
C000 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C010 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C020 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C030 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C040 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C050 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C060 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
C070 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
h〕
```

10-3-4　ディスクへデータを書き込む(Write data)

ディスクへデータを書き込むコマンドは3つあります。

1)コマンド1とコマンド17

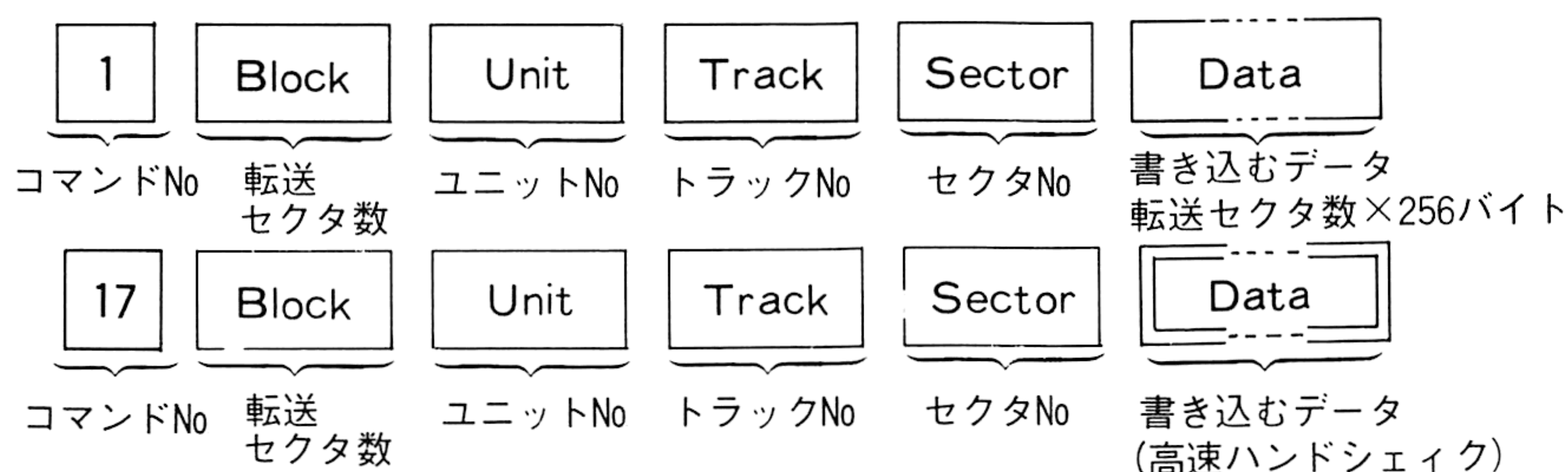
メインシステムから送られたデータをディスクへ書き込むコマンドです。(コマンド17は高速ハンドシェイク版です。)メインシステムから送られたデータをいったんライトバッファ(4000H～4FFFH)に置き、それを指定したセクタへ書き込みます。転送セクタ数、ユニット番号、トラック番号、セクタ番号を指定します。

これらは、コマンド2の場合と同じです。

正常に書き込めなかった場合には、コマンドステータスのIOERRフラグが1になります。

コマンドフォーマット

1	Block	Unit	Track	Sector	Data
コマンドNo	転送セクタ数	ユニットNo	トラックNo	セクタNo	書き込むデータ 転送セクタ数×256バイト

17	Block	Unit	Track	Sector	Data
コマンドNo	転送セクタ数	ユニットNo	トラックNo	セクタNo	書き込むデータ (高速ハンドシェイク)

(転送セクタ数)+(セクタNo)≦17

例文として、メインシステムのC000Hからのデータをディスクへ書き込むプログラムを載せておきます。

リスト10-8

```
100 GOSUB *INTHS
110 INPUT "Input drive  number (1,2)  ";DRIVE
120 IF DRIVE<>1 AND DRIVE<>2 THEN 110
130 INPUT "Input track  number (0-79) ";TRACK
140 IF TRACK<0 OR TRACK>79 THEN 130
150 INPUT "Input sector number (1-16) ";SECTOR
160 IF SECTOR<1 OR SECTOR>16 THEN 150
170 I'JPUT "Input block  number (1-16) ";BLOCK
180 IF BLOCK<1 OR BLOCK>16 OR SECTOR+BLOCK>17 THEN 170
190 BUFF=&HC000
200 A=1         :GOSUB *SNDCOM  コマンド1
210 A=BLOCK     :GOSUB *SNDPAR  転送セクタ数指定
220 A=DRIVE-1   :GOSUB *SNDPAR  ドライブユニット番号
230 A=TRACK     :GOSUB *SNDPAR  トラック番号
240 A=SECTOR    :GOSUB *SNDPAR  セクタ番号
250 FOR I=0 TO BLOCK*256-1              } 書き込むデータを送る。
260   A=PEEK(BUFF+I):GOSUB *SNDPAR      }
270 NEXT
300 '
310 A=6 :GOSUB *SNDCOM  } コマンド6でエラーのチェック
320      GOSUB *REVPAR  }
330 IF (A AND 1)=1 THEN PRINT "Disk I/O error":BEEP:END
340 PRINT "Complete"
350 END
```

※1000～1220:共通サブルーチン(182P)

コマンド1実行例

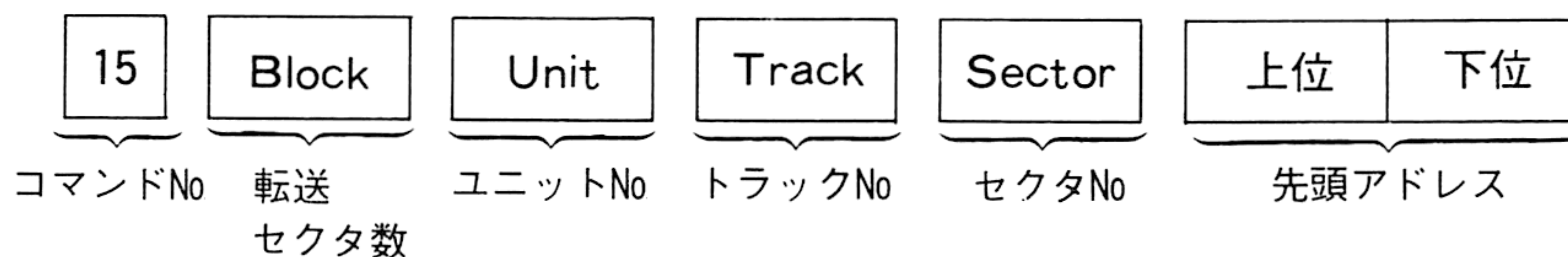
```
mon

h) dc000
C000 30 31 32 33 34 35 36 37 38 39 FF FF FF FF FF FF        0123456789
h) ^B
Ok

run
Input drive  number (1,2)  ? 1 |
Input track  number (0-79) ? 0 | C000H~C0FFHの内容をドライブ1の
Input sector number (1-16) ? 1 | トラック0、セクター1に書き込む。
Input block  number (1-16) ? 1 |
Complete
Ok

print left$(dski$(1,0,0,1),10) | 確かに書き込まれている。
0123456789
Ok
```

2)コマンド15

サブシステムの任意のアドレスから始まるデータをディスクへ書き込みます。転送セクタ数、ユニット番号、トラック番号、セクタ番号、先頭アドレスを指定します。

正常に書き込めなかった場合には、コマンドステータスのIOERRフラグが1になります。

コマンドフォーマット

15	Block	Unit	Track	Sector	上位	下位
コマンドNo	転送セクタ数	ユニットNo	トラックNo	セクタNo	先頭アドレス	

例文として、サブシステムの6000H番地からのデータをディスクへ書き込むプログラムを載せておきます。

リスト10-9

```
100 GOSUB *INTHS
110 INPUT "Input drive  number (1,2)  ";DRIVE
120 IF DRIVE<>1 AND DRIVE<>2 THEN 110
130 INPUT "Input track  number (0-79) ";TRACK
140 IF TRACK<0 OR TRACK>79 THEN 130
150 INPUT "Input sector number (1-16) ";SECTOR
160 IF SECTOR<1 OR SECTOR>16 THEN 150
170 INPUT "Input block  number (1-16) ";BLOCK
180 IF BLOCK<1 OR BLOCK>16 OR SECTOR+BLOCK>17 THEN 170
190 '
200 BUFF=&H6000:BUFF$=RIGHT$("0000"+HEX$(BUFF),4)
210 '
220 A=15                            :GOSUB *SNDCOM |コマンド15
230 A=BLOCK                         :GOSUB *SNDPAR |転送セクタ数
240 A=DRIVE-1                       :GOSUB *SNDPAR |ドライブユニット番号
250 A=TRACK                         :GOSUB *SNDPAR |トラック番号
260 A=SECTOR                        :GOSUB *SNDPAR |セクタ番号
270 A=VAL("&H"+LEFT$(BUFF$,2))      :GOSUB *SNDPAR |バッファの先頭のアドレス上位
280 A=VAL("&H"+RIGHT$(BUFF$,2))     :GOSUB *SNDPAR |      〃              下位
300 '
310 A=6 :GOSUB *SNDCOM |
320      GOSUB *REVPAR | エラーのチェック
330 IF (A AND 1)=1 THEN PRINT "Disk I/O error":BEEP:END
340 PRINT "Complete"
350 END
```

※1000~1220:共通サブルーチン(182P)

コマンド15実行例

```
load "com9.exp"
Ok
run

Input first address ? 6000    コマンド9の例を使い
Input end   address ? 6005    6000H～6005Hの内容を見る
6000 : 41,42,43,44,45,46
Ok
```

```
load "con15.exp"
Ok
run
Input drive  number (1,2)   ? 1   バッファを6000H～として
Input track  number (0-79)  ? 0   ドライブ1のトラック0、セクタ1
Input sector number (1-16)  ? 1   から1セクタ書き込む
Input block  number (1-16)  ? 1
Complete
Ok
```

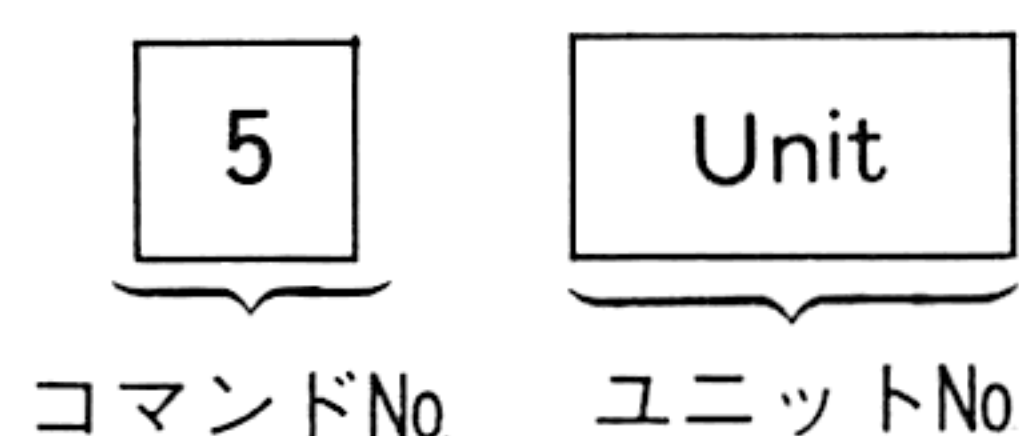
```
print left$(dski$(1,0,0,1),6)   確かに6000Hからのデータが
ABCDEF                          書き込まれている。
Ok
```

10-3-5　フォーマット(Format)

コマンド5は、ディスクの物理的フォーマットを行ないます。

フォーマットしたいディスクのユニット番号を指定すると、全トラック(両面なら80トラック、片面なら35トラック)に対し、物理的フォーマットを行ないます。フォーマットが失敗したときは、コマンドステータスのIOERRフラグが1になります。

コマンドフォーマット

5	Unit
コマンドNo	ユニットNo

リスト　10-10

```
100 GOSUB *INTHS
110 INPUT "Input drive number (1,2) ";DRIVE
120 IF DRIVE<>1 AND DRIVE<>2 THEN 110
130 PRINT "Format drive";DRIVE;:INPUT " Ok(y/n) ";DUM$
140 IF DUM$<>"y" AND DUM$<>"Y" THEN PRINT:GOTO 110
150 '
160 A=5          :GOSUB *SNDCOM  |コマンド5
170 A=DRIVE-1    :GOSUB *SNDPAR  |ドライブユニット番号
180 '
190 A=6          :GOSUB *SNDCOM  } エラーチェック
200              :GOSUB *REVPAR  }
210 IF (A AND 1)=1 THEN PRINT "Disk I/O error":BEEP:END
220 PRINT "Complete"
230 END
```

※1000～1220：共通サブルーチン(182P)

コマンド5実行例

```
run
Input drive number (1,2) ? 1
Format drive 1  Ok(y/n) ? y
Complete
Ok
```

10-3-6 コピー(Copy)

コマンド4

コマンド4は、セクタ単位でデータのコピーを取ります。

コピーするセクタ数とコピー元、コピー先のユニット、トラック、セクタ番号を指定します。コピーはライトバッファにコピー元のデータを読んできて、それをコピー先へ書き込む簡単なものです。2トラックにまたがったコピーは出来ません。

コマンドフォーマット

4	Block	Unit	Track	Sector	Unit	Track	Sector
コマンドNo	コピーセクタ数	コピー元のユニット、トラック、セクタ (source)			コピー先のユニット、トラック、セクタ (destination)		

(コピーセクタ数) ＋ (コピー元のセクタ番号) ≦17
(コピーセクタ数) ＋ (コピー先のセクタ番号) ≦17

リスト 10-11

```
100 GOSUB *INTHS
110 INPUT "Input source drive  (1,2)  ";S.DRIVE
120 IF S.DRIVE<>1 AND S.DRIVE<>2 THEN 110
130 INPUT "Input source track (0-79)  ";S.TRACK
140 IF S.TRACK<0 OR S.TRACK>79 THEN 130
150 INPUT "Input source sector (1-16) ";S.SECTOR
160 IF S.SECTOR<1 OR S.SECTOR>16 THEN 150
170 PRINT
180 INPUT "Input destination drive  (1,2)  ";D.DRIVE
190 IF D.DRIVE<>1 AND D.DRIVE<>2 THEN 180
200 INPUT "Input destination track (0-79)  ";D.TRACK
210 IF D.TRACK<0 OR D.TRACK>79 THEN 200
220 INPUT "Input destination sector (1-16) ";D.SECTOR
230 IF D.SECTOR<1 OR D.SECTOR>16 THEN 220
240 PRINT
250 INPUT "Input number of copy block (1-16) ";BLOCK
260 IF BLOCK<1 OR BLOCK>16 THEN 250
270 IF S.SECTOR+BLOCK>17 OR D.SECTOR+BLOCK>17 THEN 250
280 '
290 A=4          :GOSUB *SNDCOM  |コマンド4
300 A=BLOCK      :GOSUB *SNDPAR  |コピーするセクタ数
310 A=S.DRIVE-1  :GOSUB *SNDPAR  }
320 A=S.TRACK    :GOSUB *SNDPAR  } コピー元のドライブ、ユニット、トラック、セクタ番号
330 A=S.SECTOR   :GOSUB *SNDPAR  }
340 A=D.DRIVE-1  :GOSUB *SNDPAR  }
350 A=D.TRACK    :GOSUB *SNDPAR  } コピー先のドライブユニット、トラック、セクタ番号
360 A=D.SECTOR   :GOSUB *SNDPAR  }
370 '
380 A=6          :GOSUB *SNDCOM  } エラーのチェック
390               GOSUB *REVPAR  }
400 IF (A AND 1)=1 THEN PRINT "Disk I/O error":BEEP:END
```

```
410 PRINT "Complete"
420 END
```
※1000～1220：共通サブルーチン(182P)

コマンド4実行例

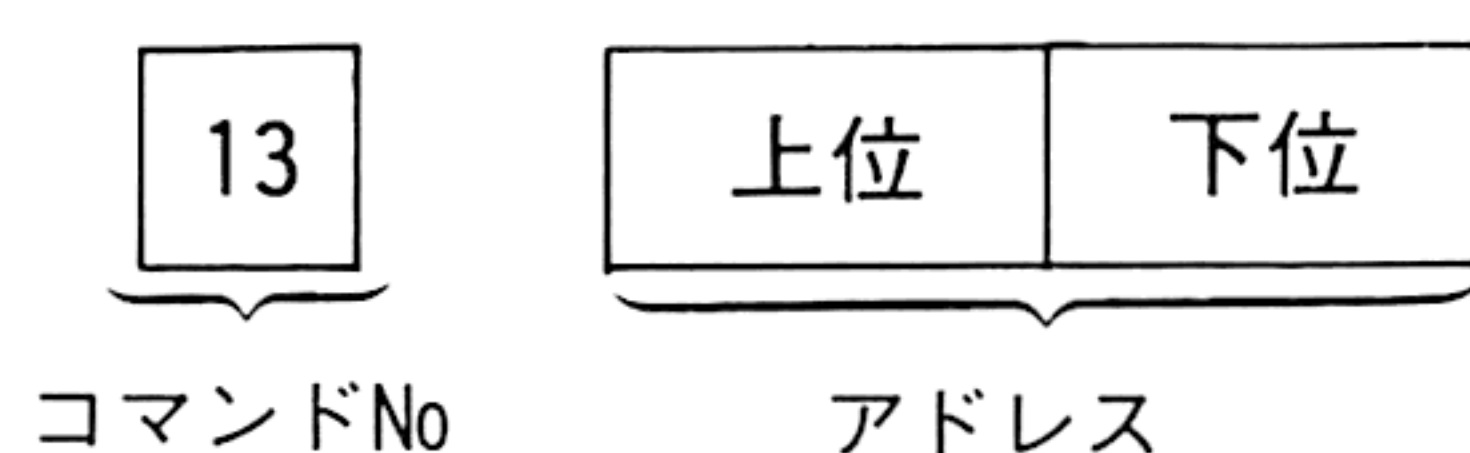
```
print left$(dski$(2,0,0,1),10);" - ";lift$(dski$(2,0,0,2),10)
0123456789 - ABCDEFGHIJ
Ok

run
Input source drive  (1,2)  ? 1 ┐
Input source track  (0-79)  ? 0 ├ コピー元指定
Input source sector (1-16) ? 1 ┘

Input destination drive  (1,2)  ? 1 ┐
Input destination track (0-79)  ? 0 ├ コピー先指定
Input destination sector (1-16) ? 2 ┘

Input number of copy block (1-16) ? 1 | 1セクタだけコピーする
Complete
Ok

print left$(dski$(2,0,0,1),10);" - ";left$(dski$(2,0,0,2),10)
0123456789 - 0123456789 | 確かにコピーされた
Ok
```

10-3-7　ユーザーのプログラムを動かす

1)コマンド13

コマンド13は、指定したアドレスへ実行を移すコマンドです。このコマンドを使えば、サブシステムの中で、自分が作ったプログラムを動かすことができます。実行を終了して、ROMのプログラムに戻るときは、JP　00C1H(C3H, C1H, 00H)とします。サブシステム内での暴走はディスクの誤動作を招く恐れがあるので、十分注意して下さい。

コマンドフォーマット

13		上位	下位
コマンドNo		アドレス	

2)コマンド16

ドライブユニット0の、トラック0セクタ1から1セクタを5000H番地に読み込み、5000Hへ実行を移すコマンドです。サブシステムのみでのオートスタートに使うコマンドなので、あまり役にたちません。

コマンドフォーマット

16
コマンドNo

3)コマンド27

ブレークポイントを使ったデバッグのとき、プログラムを実行させるコマンドです。10-3-11を見て下さい。

10-3-8　ステータス情報に関するコマンド

1)コマンドステータス(コマンド6)

コマンドステータスには、2つの情報が含まれています。IOERRフラグ、FULLフラグと呼ばれるもので、IOERRフラグはディスクをアクセスするコマンド(リードデータ、ライトデータ、フォーマット、コピーコマンド)を実行した時、成功すれば0、失敗すれば1になります。メインシステム側では、IOERRフラグが1のとき、Disk I/O Errorが起きたと判断します。

FULLフラグは、コマンド2を実行してエラーが起きず、データがリードバッファにあるとき1になり、その後コマンド3でメインシステムへデータを送り終えると0になります。また、他のディスクをアクセスするコマンドを使うと、FULLフラグは0になります。

コマンドフォーマット

6	コマンドステータス
コマンドNo.	

7	6	5	4	3	2	1	0
DONE	FULL	╳	╳	╳	╳	╳	IO ERR

IOERR.....処理中にI/Oエラーが起きたら1そうでなければ0

FULL......データ・アベイラブル・フラグ

　　リード・バッファにデータがあるとき1そうでないとき0

DONE......ラストI/Oコンプリート・フラグ

　　DISKがインテリジェント方式なので常に1

リスト　10-12

```
100 GOSUB *INTHS
110 A=6 :GOSUB *SNDCOM
120      GOSUB *REVPAR
130 PRINT "Command status = ";
140 FOR BIT=7 TO 0 STEP -1
150   IF (A AND 2^BIT)=0 THEN PRINT "0 "; ELSE PRINT "1 ";
160 NEXT
170 PRINT:PRINT
180 PRINT "I/O complete     flag = ";
190 IF (A AND 2^7)=0 THEN PRINT "0" ELSE PRINT "1"
200 PRINT "Full (read buff) flag = ";
210 IF (A AND 2^6)=0 THEN PRINT "0" ELSE PRINT "1"
220 PRINT "I/O error        flag = ";
230 IF (A AND 2^0)=0 THEN PRINT "0" ELSE PRINT "1"
240 END
```

※1000～1220：共通サブルーチン(182P)

コマンド6実行例

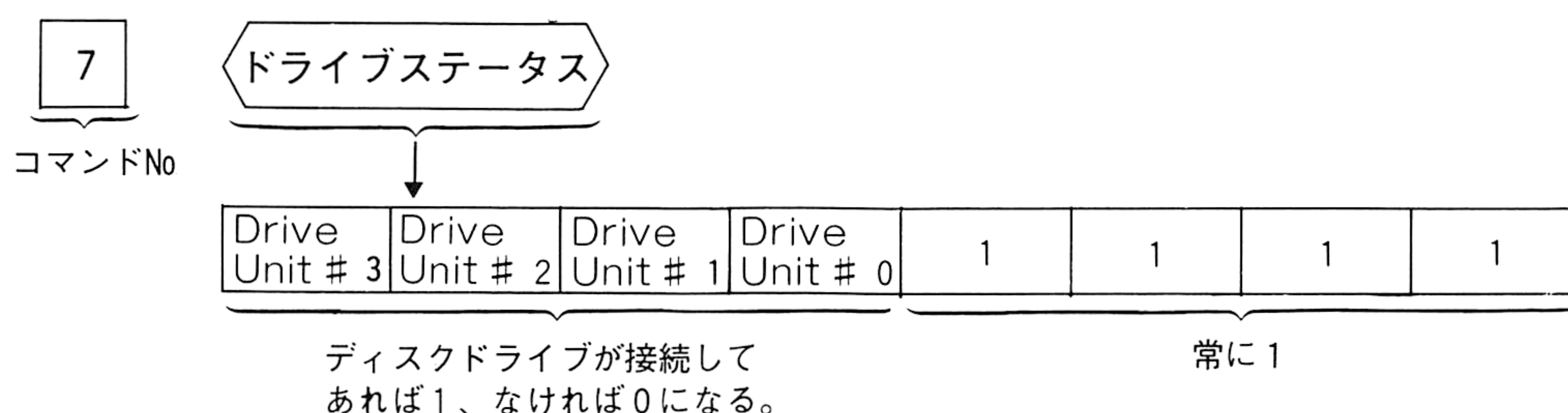
```
run
Command status = 1 0 0 0 0 0 0 0

I/O complete       flag = 1
Full (read buff)   flag = 0
I/O error          flag = 0
Ok
```

2)ドライブステータス(コマンド7)

ドライブステータスを調べれば、どのディスクドライブが接続されているかがわかります。

コマンドフォーマット

7	ドライブステータス
コマンドNo	

Drive Unit # 3	Drive Unit # 2	Drive Unit # 1	Drive Unit # 0	1	1	1	1
ディスクドライブが接続してあれば1、なければ0になる。				常に1			

ドライブステータスが1FHなら1台、3FHなら2台，FFHなら4台のディスクドライブが接続されていることになります。PC-8801mkⅡでは、起動時にドライブ数を調べて、メインシステムのワークエリア、EF62H番地にセットするようになっています。

リスト 10-13

```
100 GOSUB *INTHS
110 A=7 :GOSUB *SNDCOM
120      GOSUB *REVPAR
130 PRINT "Drive status = ";
140 FOR BIT=7 TO 0 STEP -1
150   IF (A AND 2^BIT)=0 THEN PRINT "0 "; ELSE PRINT "1 ";
160 NEXT
170 PRINT:PRINT
180 FOR DRIVE=1 TO 4
190   PRINT "Drive";DRIVE;" : ";
200   IF (A AND 2^(DRIVE+3))=0 THEN PRINT "Not connect" ELSE PRINT "Connect"
210 NEXT
220 END
```

※1000～1220：共通サブルーチン(182P)

コマンド7実行例

```
run
Drive status = 0 0 1 1 1 1 1 1

Drive 1  : Connect
Drive 2  : Connect
Drive 3  : Not connect
Drive 4  : Not connect
Ok
```

3）デバイスステータス（コマンド20）

主にライトプロテクトシールの有無を調べるためのコマンドです。

コマンドフォーマット

ビット番号	ステータス名称	略称	内容
D 7	Fault	FT	デバイスからのFault信号の状態
D 6	Write Protected	WP	デバイスからのWrite Protected信号の状態
D 5	Ready	RY	デバイスからのReady信号の状態
D 4	Track 0	T 0	デバイスからのTrack 0信号の状態
D 3	Two Side	TS	デバイスからのTwo Side信号の状態
D 2	Head Address	HD	デバイスへのSide Select信号の状態
D 1	Unit Select 1	US 1	デバイスへのUnit Select 1信号の状態
D 0	Unit Select 0	US 0	デバイスへのUnit Select 0信号の状態

指定したドライブのデバイスステータスのビット6（Write Protected）が1なら、そのドライブに入っているディスクにライトプロテクトシールが貼ってあることになります。

リスト 10-14

```
100 GOSUB *INTHS
110 INPUT "Input drive number (1-4) ";DRIVE
120 IF DRIVE<1 OR DRIVE>4 THEN 110
130 A=20          :GOSUB *SNDCOM
140 A=DRIVE-1     :GOSUB *SNDPAR
150                GOSUB *REVPAR
160 PRINT:PRINT "Device status = ";
170 FOR BIT=7 TO 0 STEP -1
180   IF (A AND 2^BIT)=0 THEN PRINT "0 "; ELSE PRINT "1 ";
190 NEXT
200 PRINT:PRINT
210 PRINT "Unit select   =";A AND 3
220 PRINT "Head address  = ";
230 IF (A AND 2^2)=0 THEN PRINT "0" ELSE PRINT "1"
240 PRINT "Two side line = ";
250 IF (A AND 2^3)=0 THEN PRINT "0" ELSE PRINT "1"
260 PRINT "Track 0 line  = ";
270 IF (A AND 2^4)=0 THEN PRINT "0" ELSE PRINT "1"
280 PRINT "Ready line    = ";
290 IF (A AND 2^5)=0 THEN PRINT "0" ELSE PRINT "1"
300 PRINT "Write protect = ";
310 IF (A AND 2^6)=0 THEN PRINT "0" ELSE PRINT "1"
320 PRINT "Fault line    = ";
330 IF (A AND 2^7)=0 THEN PRINT "0" ELSE PRINT "1"
340 END
```

※1000～1220：共通サブルーチン（182P）

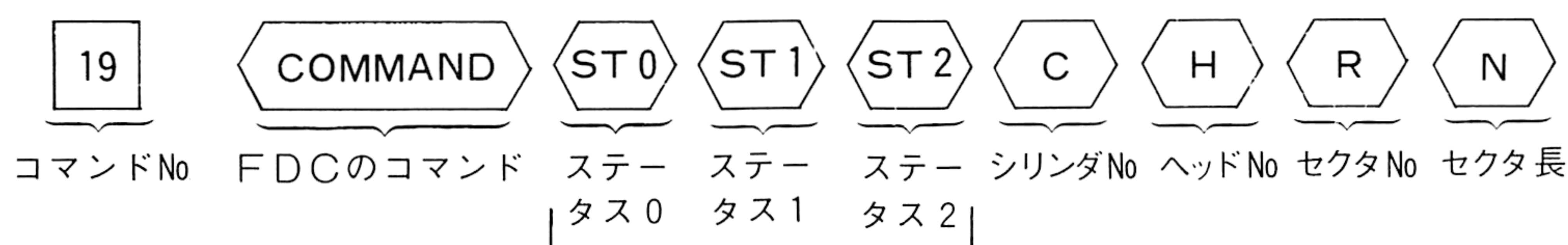
コマンド20実行例

```
run
Input drive number (1-4) ? 2

Device status = 0 0 1 1 1 0 0 1

Unit select   = 1
Head address  = 0
Two side line = 1
Track 0 line  = 1
Ready line    = 1
Write protect = 0
Fault line    = 0
Ok
```

4)リザルトステータス(コマンド19)

フロッピーディスクコントローラ(FDC, μPD765)のコマンドの実行結果を送ります。送られてくる情報は、実行したコマンド番号、ステータスレジスタ0、ステータスレジスタ1、ステータスレジスタ2、実行終了セクタのシリンダ、ヘッド、セクタ、セクタ長の8つです。

コマンドフォーマット

19	COMMAND	ST0	ST1	ST2	C	H	R	N
コマンドNo	FDCのコマンド	ステータス0	ステータス1	ステータス2	シリンダNo	ヘッドNo	セクタNo	セクタ長

(ステータス0〜2)内容については付録00P参照

リスト　10-15

```
100 GOSUB *INTHS
110 A=19:GOSUB *SNDCOM
120     GOSUB *REVPAR:PRINT "command code = ";RIGHT$("00"+HEX$(A),2)
130     PRINT
140     GOSUB *REVPAR:PRINT "Status0 = ";:GOSUB *DSPBIT:
150     GOSUB *REVPAR:PRINT "Status1 = ";:GOSUB *DSPBIT
160     GOSUB *REVPAR:PRINT "Status2 = ";:GOSUB *DSPBIT
170     PRINT
180     GOSUB *REVPAR:PRINT "# of cylinder =";A
190     GOSUB *REVPAR:PRINT "# of head     =";A
200     GOSUB *REVPAR:PRINT "# of sector   =";A
210     GOSUB *REVPAR:PRINT "Sector length =";A
220 END
230 *DSPBIT
240 FOR BIT=7 TO 0 STEP -1
250   IF (A AND 2^BIT)=0 THEN PRINT "0 "; ELSE PRINT "1 ";
260 NEXT
270 PRINT
280 RETURN
```

※1000〜1220:共通サブルーチン(182P)

コマンド19実行例

```
a$=dski$(1,0,0,2)
Ok

run
command code = 46

Status0 = 0 0 0 0 0 0 0 0
Status1 = 0 0 0 0 0 0 0 0
Status2 = 0 0 0 0 0 0 0 0

# of cylinder = 1
# of head     = 0
# of sector   = 3
Sector length = 1
Ok
```

10-3-9　両面、片面モードの指定

1)コマンド23

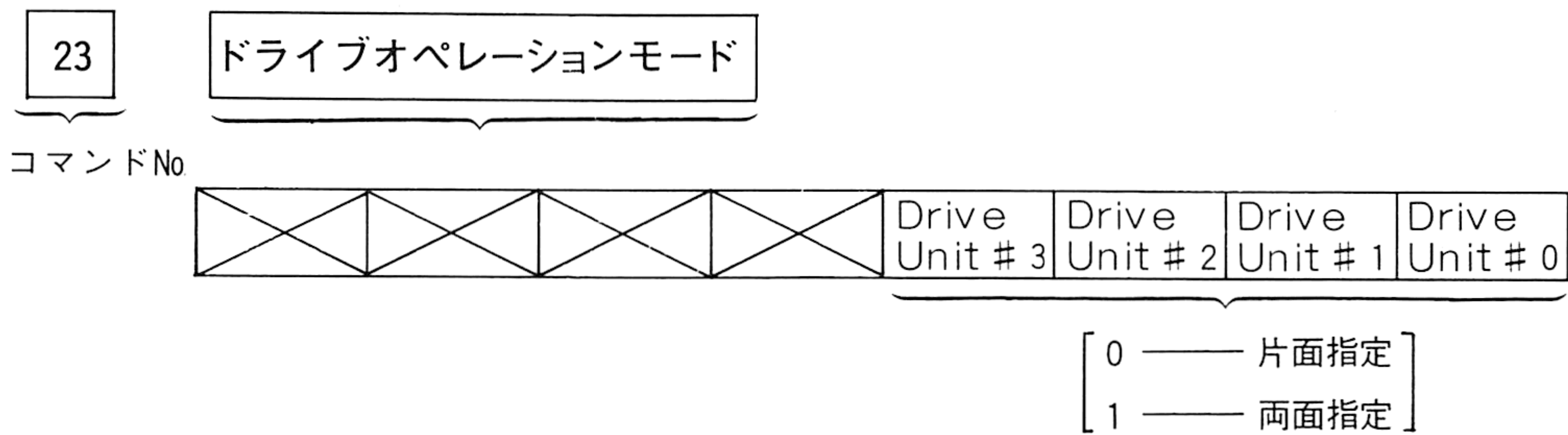
各ドライブごとに両面、片面の指定を行ないます。このコマンドは5インチ片面ディスクPC-8031／1W／1Vとのコンパチビリティーを保つためのもので、サブシステム自身の起動時はすべて片面モードになっています。メインシステムでは、サブシステムROMの07EFHがEFHのとき、5インチ両面ドライブが接続されていると判断し、コマンド23ですべてのドライブを両面モードに切り換えます。また、07EFHがFFHなら、5インチ片面ドライブが接続されていると判断します。

コマンドフォーマット

23	ドライブオペレーションモード

コマンドNo

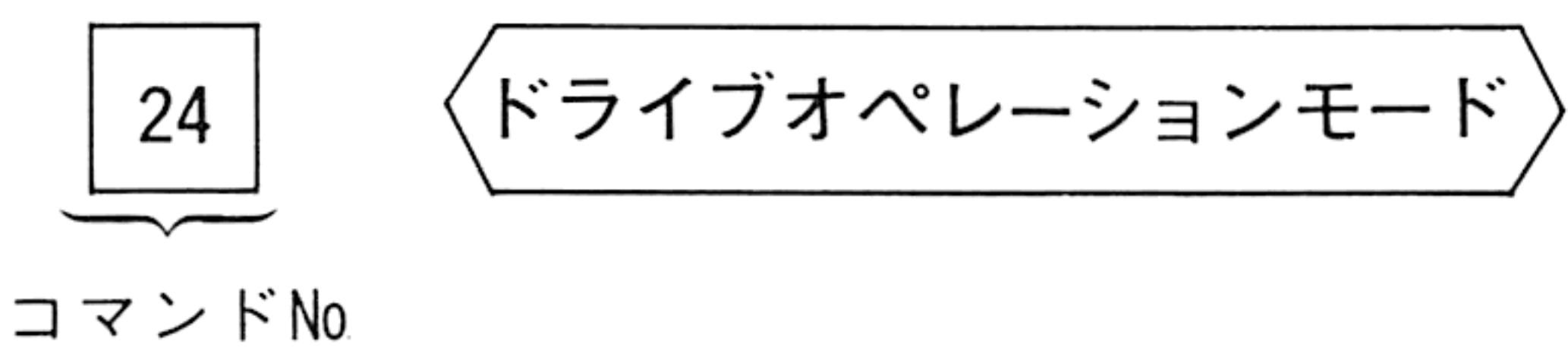
╳	╳	╳	╳	Drive Unit ＃3	Drive Unit ＃2	Drive Unit ＃1	Drive Unit ＃0

0 ―― 片面指定
1 ―― 両面指定

2)コマンド24

現在の両面・片面指定の情報、ドライブオペレーションモードを送ってきます。

コマンドフォーマット

24	ドライブオペレーションモード

コマンドNo

リスト　10-16

```
100 GOSUB *INTHS
110 A=24:GOSUB *SNDCOM
120       GOSUB *REVPAR
130 PRINT "Drive operation mode = ";
140 FOR BIT=7 TO 0 STEP -1
150   IF (A AND 2^BIT)=0 THEN PRINT "0 "; ELSE PRINT "1 ";
160 NEXT
170 PRINT:PRINT
180 FOR DRIVE=1 TO 4
190   PRINT "Drive";DRIVE;" : ";
200   IF (A AND 2^(DRIVE-1))=0 THEN PRINT "Single" ELSE PRINT "Double"
210 NEXT
220 END
```

※1000～1220：共通サブルーチン(182P)

コマンド24実行例

```
run
Drive operation mode = 0 0 0 0 1 1 1 1

Drive 1  : Double
Drive 2  : Double
Drive 3  : Double
Drive 4  : Double
Ok
```

10-3-10　リードアフターライト

リードアフターライト(Read　after　write)とは、データをディスクへ書き込んだ後、正しく書き込まれたかどうか、チェックすることです。リードアフターライトモードになると、データの書き込み終了後、実際にデータを読んできて、書き込むデータと比較します。これらが違っていれば再度、書き込みを行ないます。

このモードでのデータの書き込みは、そうでないときより2～3倍ぐらい遅くなります。

1)コマンド25

リードアフターライトモードにします。

コマンドフォーマット

25

コマンドNo

2)コマンド26

リードアフターライトモードを解除します。

コマンドフォーマット

26

コマンドNo

リスト 10-17

```
100 GOSUB *INTHS
110 INPUT "Set read after write flag (y/n) ";DUM$
120 IF DUM$="y" OR DUM$="Y" THEN 200
130 INPUT "Reset read after write flag , OK (y/n) ";DUM$
140 IF DUM$<>"y" AND DUM$<>"Y" THEN 110
150 '
160 A=26 : GOSUB *SNDCOM   } リードアフターライトモードを解除
170 PRINT "Complete"       }
180 END
200 '
210 A=25 : GOSUB *SNDCOM   } リードアフターライトモード
220 PRINT "Complete"       }
230 END
```

※1000～1220：共通サブルーチン(182P)

コマンド25実行例

```
run
Set read after write flag (y/n) ? y } リードアフターライトモード
Complete
Ok

time$="00:00:00":for i=0 to 8:dsko$ 1,0,0,1:next:print time$
00:00:03
Ok
```

※リードアフターライトモードを指定すると書き込む速度は遅くなる。

```
run
Set read after write flag (y/n) ? n              }
Reset read after write flag , OK (y/n) ? y       } モード解除
Complete                                         }
Ok                                               }

time$="00:00:00":for i=0 to 8:dsko$ 1,0,0,1:next:print time$
00:00:01
Ok
```

10-3-11 ブレークポイントに関するコマンド

デバッグのために、ブレークポイントが設定できるようになっています。ブレークポイントとは一種のトラップ(わな)で、マシン語の実行をそこで止めることができます。その時の各レジスタの値や、ブレークポイントのアドレスまでは正常に動作した、などの情報によりデバッグを行なうのです。ブレークポイントに指定したアドレスには、RST 08H(0CFH)命令がセットされます。プログラムの実行中にこの命令を通ると、0008Hに実行が移り、各レジスタの値をワークエリアに保存した後、ROMプログラムのホットスタート(00C1H)に行くようになっています。ですからRST 08H命令が実行されないと意味を持ちません。

(例1)

```
6000：21, 00, 00, C9          (LD  HL, 0000H : RET)
         ↑
```

ブレークポイントを6001Hに指定すると

```
6000：21, CF, 00, C9          (LD  HL, 00CFH : RET)
```

6000H番地から実行させても止まらない。

(例２)

6000：21, 00, 00, C9　　　　　(LD　HL, 0000H　：RET)

↑

ブレークポイントを6000Hに指定すると

6000：CF, 00, 00, C9　　　　　(RST　08H　：……)

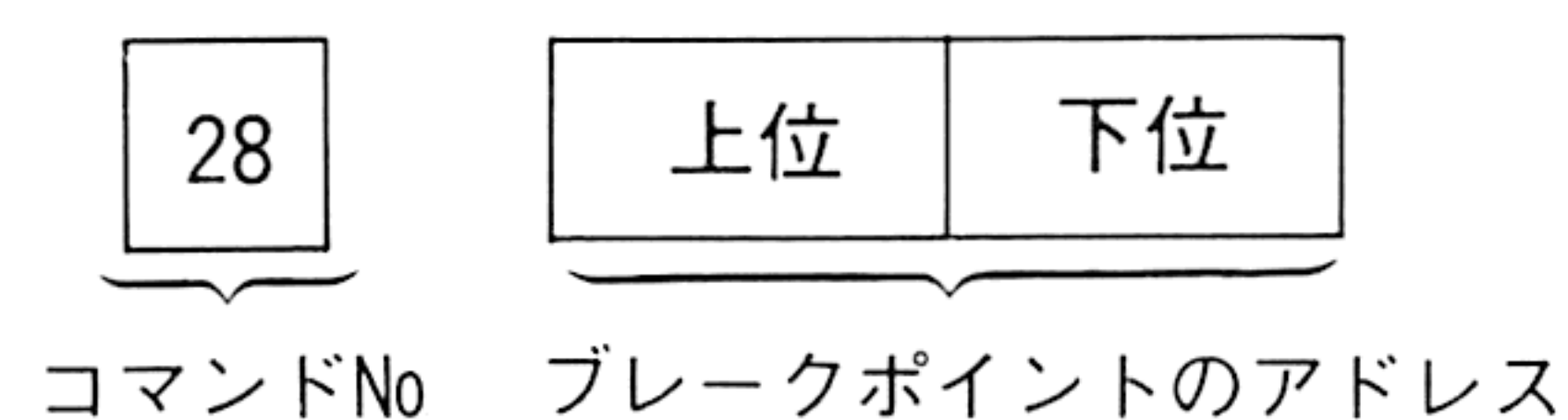
6000H番地から実行させると止まる。

ブレークポイント用のコマンドは４つあります。

１)コマンド28(ブレークポイントのアドレスを決める)

前回指定していたら、そのブレークポイントを元に戻して、指定したアドレスへRST 08Hを置き、ブレーク・ポイントを設定します。

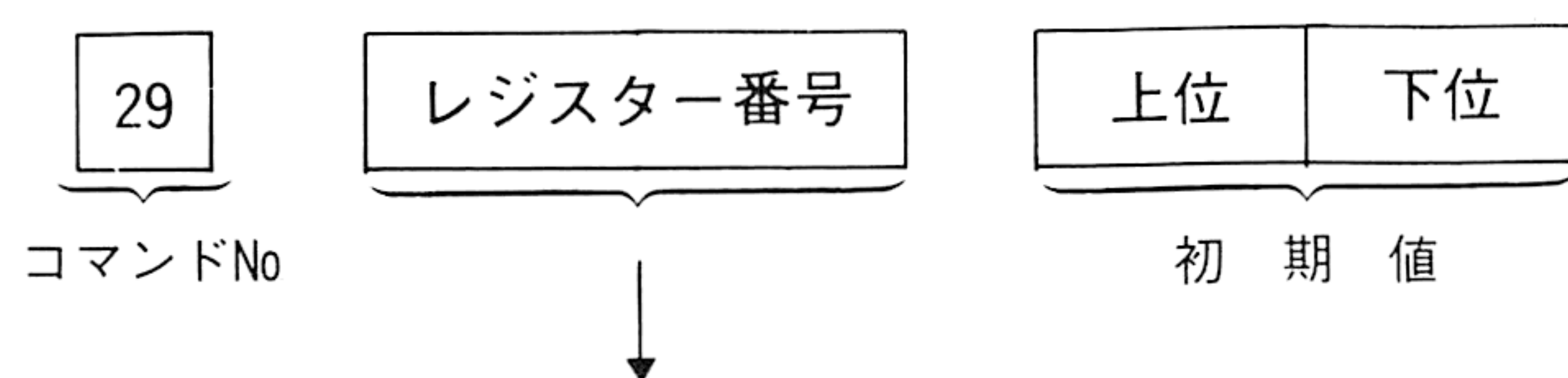
コマンドフォーマット

28		上位	下位
コマンドNo		ブレークポイントのアドレス	

２)コマンド29(各レジスタの初期値を変える)

ブレークポイントによるデバッグ用のGOコマンド(ユーザーのプログラムを実行させるコマンド。コマンド27)では、各レジスタの値を自由に設定できるようになっています。7F28H～7F41Hまでに全レジスタのワークエリアがあり、それらの値を持ってPC(プログラムカウンタ)の示すアドレスへ実行を移すようになっています。

コマンド29は、このレジスタ用ワークエリアの値を自由に変えることができます。

コマンドフォーマット

29	レジスター番号	上位	下位
コマンドNo	↓	初　期　値	

レジスター番号	レジスターの名前
0	AF
1	BC
2	DE
3	HL
4	AF'
5	BC'
6	DE'

レジスター番号	レジスターの名前
7	HL'
8	IX
9	IY
10	I　*)
11	PC
12	SP

*)　Ｉレジスタは下位８ビットのみ使用

3)コマンド27(デバッグするプログラムを実行する)

コマンド29で決められたPC(プログラムカウンタ)のアドレスへ実行を移します。

コマンドフォーマット

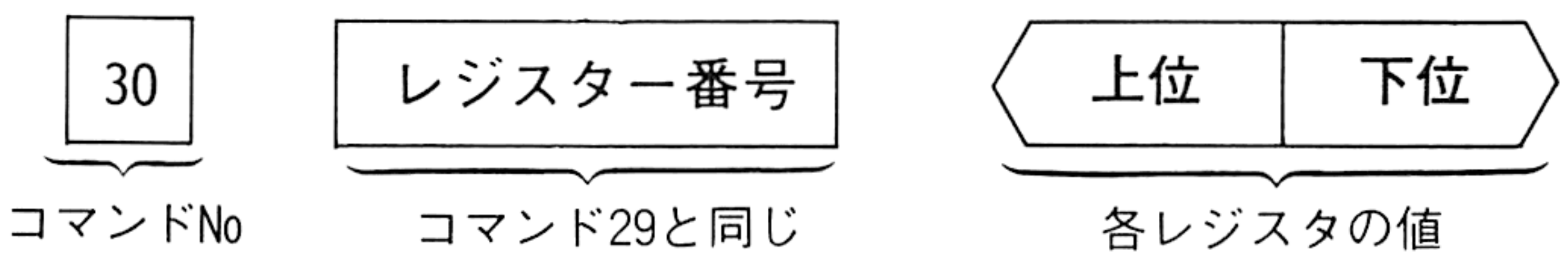
27

コマンドNo

4)コマンド30(ブレークポイントで止まったときのレジスタの値を見る)

ブレークポイント(RST 08H)で実行が中止させられると、各レジスタの値をワークエリア(7F28H～7F41H)に退避するようになっています。コマンド30は、このレジスタ用ワークエリアの値をみることができます。

コマンドフォーマット

30	レジスター番号	上位	下位
コマンドNo	コマンド29と同じ	各レジスタの値	

レジスタ用ワークエリア

7F28 ――― SP	7F36 ――― BC’
7F2A ――― PC	7F38 ――― AF’
7F2C ――― I	7F3A ――― HL
7F2E ――― IY	7F3C ――― DE
7F30 ――― IX	7F3E ――― BC
7F32 ――― HL’	7F40 ――― AF
7F34 ――― DE’	

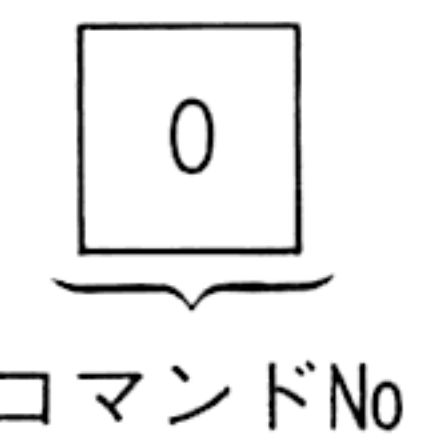
10- 3 -12 その他

1)コマンド 0 (ディスクのイニシャライズ)

サブシステムの初期化。このコマンドは、起動時(コールドスタート、ホットスタートのとき)に1回だけ行なわれます。

コマンドフォーマット

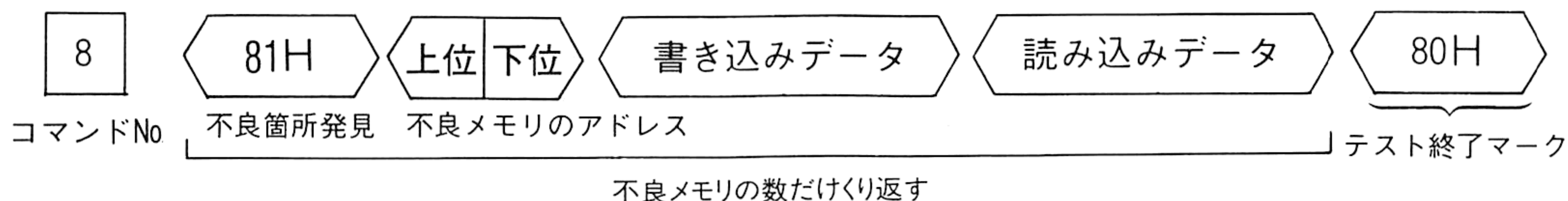
0

コマンドNo

2)**コマンド8**(メモリーテスト)

4000H～7EFFHまでのRAMのテストを行ないます。方法としては、1バイトごとに0～255のデータを書き込み、正しく書かれたかどうかをチェックします。もし、エラーが発生したなら、その部分のRAMは交換しなければなりません。このテストは2分程かかります。

コマンドフォーマット

8	81H	上位	下位	書き込みデータ	読み込みデータ	80H
コマンドNo	不良箇所発見	不良メモリのアドレス				テスト終了マーク

(81H～読み込みデータ：不良メモリの数だけくり返す)

リスト 10-19

```
100 GOSUB *INTHS
110 A=8 :GOSUB *SNDCOM
120       GOSUB *REVPAR
130 IF A=&H80 THEN PRINT "Complete":END
140       GOSUB *REVPAR:ADD=A*256
150       GOSUB *REVPAR:ADD=ADD+A
160       GOSUB *REVPAR:WRITE.DATA=A
170       GOSUB *REVPAR:READ.DATA=A
180 PRINT:PRINT "Found bad address !":BEEP
190 PRINT "Address    = ";RIGHT$("0000"+HEX$(ADD),4)
200 PRINT "Write data = ";RIGHT$("00"+HEX$(WRITE.DATA),2)
210 PRINT "Read  data = ";RIGHT$("00"+HEX$(READ.DATA),2)
220 GOTO 120
```

※1000～1220:共通サブルーチン(182P)

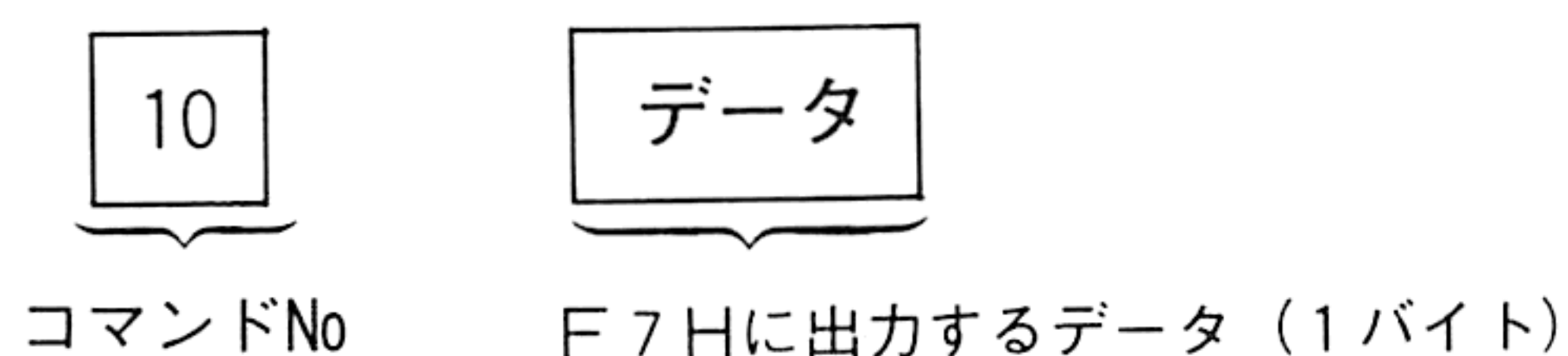
コマンド8実行例

```
run
Complete
Ok
```

3)コマンド10

I/OアドレスF7Hにデータを出力します。このコマンドは使いません。

コマンドフォーマット

10	データ
コマンドNo	F7Hに出力するデータ(1バイト)

10-4　ユーティリティ

10-4-1　サブシステム用タイニーモニタ

サブシステムのメモリを扱うモニタプログラムです。サブシステムの各種コマンドを使い、メモリのエディットやブレークポイントの設定ができます。

リスト　10-20

```
;----------------------------------------------------------------------------;
; Monitor of sub_system ( 5'disk system )  (Command mode)    (Edit mode)    ;
;----------------------------------------------------------------------------;
;- Edit  First:7000                                                          ;
;-----------------------------------------------------------;----------------;
; XX00 - 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ; (CODE) - (ASC) ;
;-----------------------------------------------------------;----------------;
; 7000 - C3 12 70 C3 33 70 00 40 00 02 01 00 00 01 00 00 ;テ.pテ3p.@.........;
; 7010 - 01 01 CD D6 70 CD F4 70 CD 59 70 DA 53 70 3E 80 ;..ヘヨpヘ日pヘYpレSp>_;
; 7020 - DF 2A 06 70 ED 4B 08 70 7E DF 23 0B 79 B0 20 F8 ;゜*.p○K.p~゜#.y- .;
; 7030 - C3 C1 00 CD D6 70 CD F4 70 2A 06 70 ED 4B 08 70 ;テチ.ヘヨpヘ日p*.p○K.p;
; 7040 - D7 77 23 0B 79 B0 20 F8 CD 94 70 38 06 3E 80 DF ;ラw#.y- .ヘ‾p8.>_゜;
; 7050 - C3 C1 00 3E 81 DF C3 C1 00 CD 0F 71 D8 CD 9D 71 ;テチ.>▂゜テチ.ヘ.qリヘ、q;
; 7060 - 3E 09 32 0A 7F 3E 46 CD A4 02 CD 5A 71 2A 06 70 ;>.2. >Fヘ、.ヘZq*.p;
; 7070 - ED 4B 08 70 FB 76 DB FA E6 20 28 09 DB FB 77 23 ;○K.p.vロ.◥ (.ロ.w#;
; 7080 - 0B 79 B0 20 EF CD 77 71 D0 CD B0 71 21 0A 7F 35 ;.y- ＼ヘwqミヘーq!. 5;
; 7090 - 20 D3 37 C9 CD 0F 71 D8 CD 9D 71 3E 09 32 0A 7F ; モ7ノヘ.qリヘ、q>.2. ;
; 70A0 - 3E 45 CD A4 02 CD 5A 71 2A 06 70 ED 4B 08 70 FB ;>Eヘ、.ヘZq*.p○K.p.;
; 70B0 - 76 DB FA E6 20 28 09 7E D3 FB 23 0B 79 B0 20 EF ;vロ.◥ (.~モ.#.y- ＼;
; 70C0 - CD 77 71 D0 3A 0E 7F E6 02 37 C0 CD B0 71 21 0A ;ヘwqミ:. ◥.7タヘーq!.;
; 70D0 - 7F 35 20 CC 37 C9 D7 32 0A 70 57 D7 32 0B 70 32 ; 5 フ7ノラ2.pWラ2.p2;
; 70E0 - 0E 70 D7 32 0C 70 32 0F 70 07 07 B2 32 0D 70 D7 ;.pラ2.p2.p..イ2.pラ;
; 70F0 - 32 10 70 C9 21 DA 71 16 00 3E 02 07 5F 19 5E 23 ;2.pノ!レq..>.._.^#;
;----------------------------------------------------------------------------;
```

モニタのコマンドは10個あり、各コマンドの頭文字を入力すると、そのコマンド名を表示して、次のパラメータの入力を待つように作ってあります。コマンドの入力を止めたいときは[HELP]キーを押すと初期状態に戻ります。

注)ディスクユニットのモーターが止まっていると、ハンドシェイク可能になるまで、プログラムが2秒ほど止まります。

(1)Eコマンド……メモリのエディット

コマンドモードでEを押すとエディットするメモリの先頭アドレスを尋ねてきます。

-Edit　First：6000[⏎]

例えば6000と入力して下さい。打ち間違った場合は[DEL]キーで訂正できます。

6000と入力し、そこで[⏎]キーかスペースキーを押すと、6000H～60FFHの内容が表示されます。カーソルの位置で0～9，A～Fの16進データを入力すると、そのデータがメモリ上に書き込まれます。カーソルの移動はカーソルキーで行なえ、エディットを終了するときは[⏎]キーを押します。

また、アスキーコードによる入力もできます。[HELP]キーを押して下さい。カーソルはアスキー文字列を表示している欄に移動します。そこでキーボードより、なにか文字を打ち込めば、そのアスキコードが書き込まれます。

16進入力に戻るときは再度[HELP]キーを押します。

(2)Tコマンド……メイン、サブシステム間のデータ転送

①メインシステムからサブシステムへデータを転送する場合は、Tに続いてMを押します。

-Transport　[Main ＝＞ Sub system]　 Start：C000　End：C0FF　Dest：6000⏎

転送するデータの先頭、終了アドレスと転送先(サブシステム)の先頭アドレスを指定します。アドレスの入力はエディットコマンドのときと同じです。

C0FF
C000
60FF
6000
メインシステム　　サブシステム

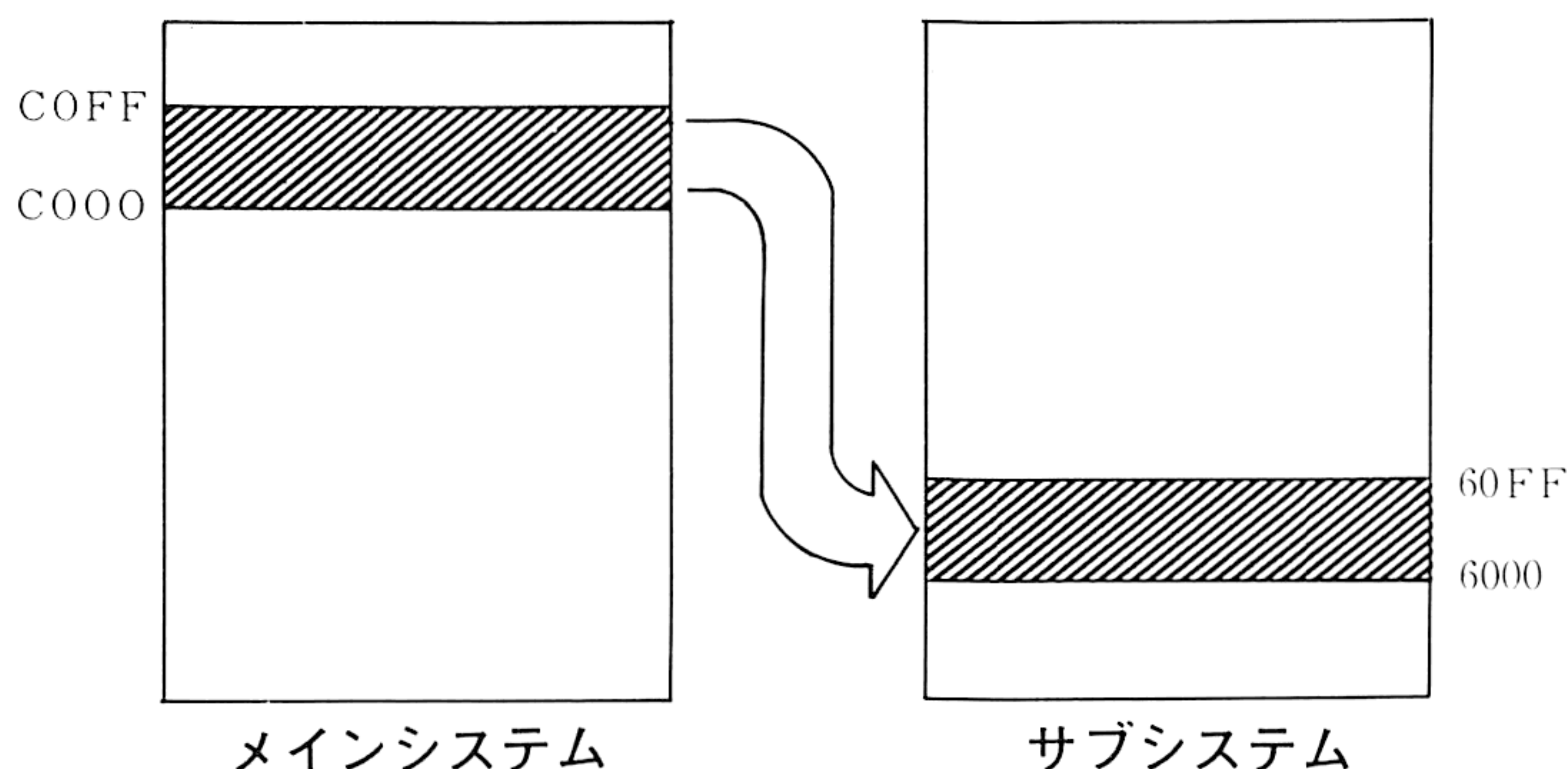



図10－4－A

②サブシステムのデータをメインシステムへ送る場合はTとSを押します。

-Transport　[Sub＝＞Main System]　Start：6000　End：60FF　Dest：C000⏎

続けてサブシステム側のデータの先頭、終了アドレスと、メインシステム側の転送する先頭アドレスを指定します。ただし、転送されるデータがメインシステムのC000H～DFFFHに入らない場合はエラーとなります。

(3)Dコマンド……プリンタへのメモリダンプ

先頭アドレスと終了アドレスを指定すると、256バイト単位でサブシステムのメモリの内容をプリンタへ出力します。

-Dump First add：6000　End add：61FF⏎

(4)Fコマンド……サブシステムのメモリを指定されたデータでうめる。

指定されたアドレスの間を一定のデータでうめるコマンドです。サブシステムの先頭、終了アドレスとデータ(1バイト)を指定します。

-Full First add：6000　End add：60FF　Constant：37

注)Constantは2バイト入力可能ですが、書き込まれるのは下位の1バイトのみです。

(5)Gコマンド……ユーザープログラムの実行

指定されたアドレスから実行します。サブシステムが暴走してもメインシステムには影響ありませんが、ハンドシェイクが不可能となり、次にこのモニタのコマンドを実行すると動かなくなります。このようなときはストップリセットをすると再び動きます。

-Go Jump address：6000⏎

(6)Mコマンド……サブシステムでのブロック転送

メモリのブロック転送を行ないます。転送したいデータの先頭、終了アドレスと転送先のアドレスを指定します。

-Move First add：6000　End add：60FF　Dest add：6200⏎

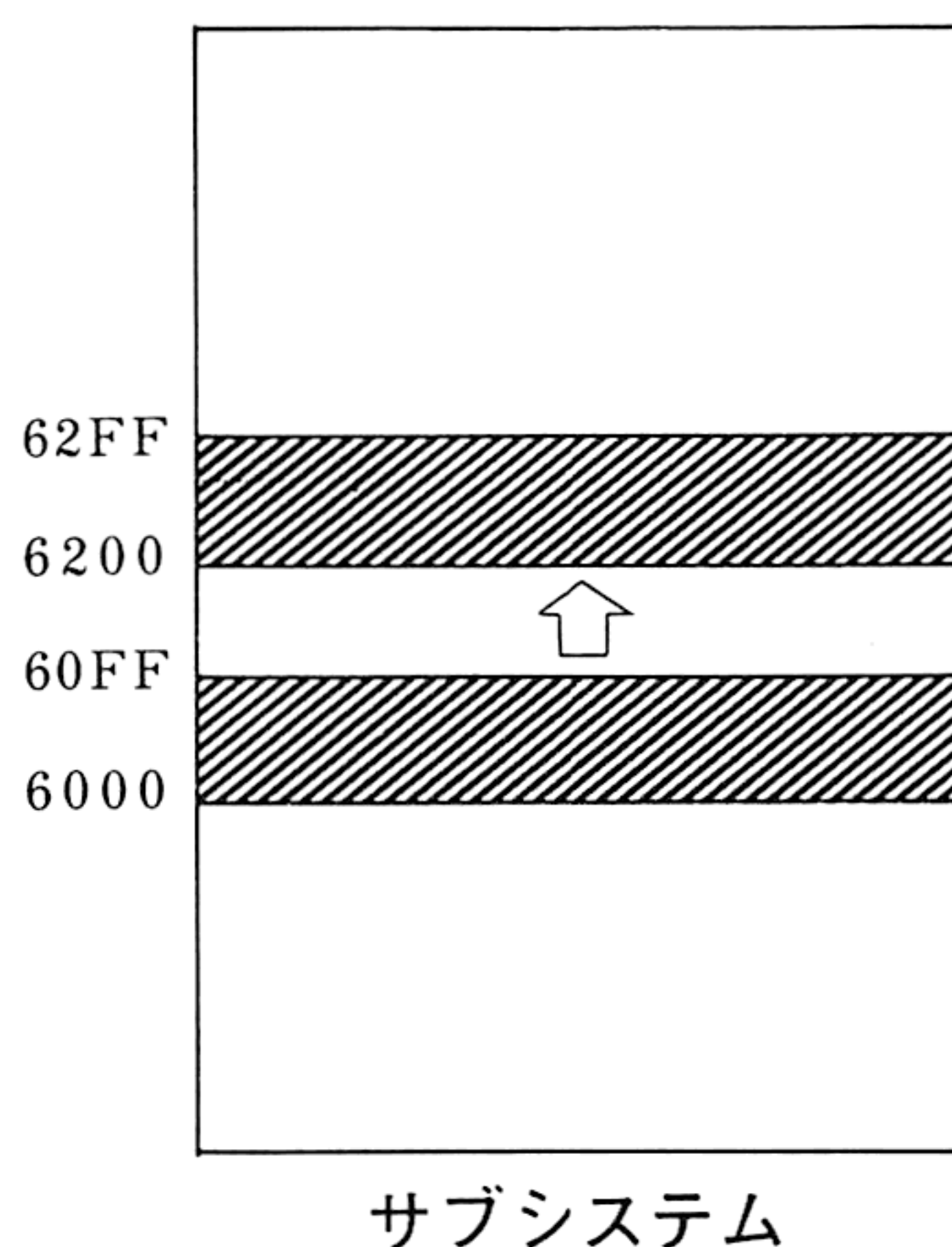
62FF
6200
60FF
6000

サブシステム

図10－４－B

(７)Rコマンド……レジスタの表示と設定

①レジスタ値の表示

サブシステムのコマンド30を使い、ブレークポイントで停止したときの各レジスタの値を表示します。Rに続いてDを押して下さい。

-Register　, dump

②レジスタ値の変更

サブシステムのコマンド29を使い、各レジスタの値を変更します。Rに続いてSを押して下さい。レジスタ名を聞いてくるので、変更したいレジスタ名を入力し、変更する値を入れて下さい。

レジスタ名は次のようになっています。

AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, I, PC, SP

-Register, set. Register name : HL'　[Now, FFFF] Value : 0101⏎

(８)Bコマンド……ブレークポイントの設定

指定したアドレスにブレークポイントを設定します。

-Set Break point　Break point : 6000⏎

(９)Cコマンド……PCレジスタより実行する。

レジスタセットで設定されたPC(プログラムカウンタ)より実行を再開します。

-Continue (Now, program counter is FFFF)……Push return key

(10)Qコマンド……このプログラムを終了してBASICに戻ります。

-Quit

マシン語はN_{88}-BASICのモニタで正確に打ち込んで

BSAVE　"edit.mac", &HE000, &H400

でBASICプログラムと同一のディスクへBSAVEして下さい。

マシン語プログラムの確認は次のチェックサムプログラムを使うと便利でしょう。

リスト 10-21 チェックサムプログラム

```
1000 '
1010 ' ----- Check sum program ver 1.00 -----
1020 '
1030   DIM SUM2(16)
1040   ON STOP GOSUB *PUSHSTOP:STOP ON
1050   PRINT "---- Check sum program ver 1.00 ----":PRINT
1060   INPUT "First address=";FADD$:FADD=VAL("&H"+FADD$)
1070   INPUT "End   address=";EADD$:EADD=VAL("&H"+EADD$)
1080   IF FADD>EADD THEN PRINT "Error ":PRINT:BEEP:GOTO 1060
1090   PRINT:INPUT "Output to CRT or Printer (C/P) ";DUM$
1100   IF DUM$="c" OR DUM$="C" THEN DEV$="SCRN:":GOTO 1130
1110   IF DUM$="p" OR DUM$="P" THEN DEV$="LPT1:":GOTO 1130
1120   GOTO 1090
1130   '
1140   OPEN DEV$ FOR OUTPUT AS #1
1150   FOR ADD=FADD TO EADD STEP 256
1160     GOSUB *DUMP
1170   NEXT ADD
1180   CLOSE
1190   '
1200   PRINT "Complete"
1210   END
1300 '
1310 *PUSHSTOP
1320   PRINT:PRINT "Complete"
1330   CLOSE
1340   STOP OFF
1350   END
2000 '
2010 *DUMP
2020   ERASE SUM2:DIM SUM2(16):SUM=0
2030   PRINT #1,
2040   '
2050   FOR I=0 TO 15
2060     PRINT #1, RIGHT$("0000"+HEX$(ADD+I*16),4);" : ";
2070     SUM1=0
2080       '
2090       FOR J=0 TO 15
2100         DAT=PEEK(ADD+I*16+J)
2110         SUM1=SUM1+DAT:SUM2(J)=SUM2(J)+DAT
2120         PRINT #1,RIGHT$("0"+HEX$(DAT),2)+" ";
2130       NEXT J
2140       '
2150     SUM=SUM+SUM1
2160     PRINT #1,": "+RIGHT$("0"+HEX$(SUM1),2)
2170   NEXT I
2180   '
2190   PRINT #1,STRING$(59,ASC("-"))
2200   PRINT #1," SUM : ";
2210   '
2220   FOR I=0 TO 15
2230     PRINT #1,RIGHT$("0"+HEX$(SUM2(I)),2);" ";
2240   NEXT I
2250   '
2260   PRINT #1,": ";RIGHT$("0"+HEX$(SUM),2)
2270   RETURN
```

実行例

```
run

First address=? e000
End   address=? e05f

Output to CRT or Printer (C/P) ? c

E000 : C3 98 E2 C3 91 E1 C3 B2 E1 C3 D1 E1 C3 1F E2 C3 : C4
E010 : 6C E2 C3 29 E1 C3 5D E1 C3 E9 E0 C3 6D E0 C3 17 : 92
E020 : E1 C3 57 E0 C3 43 E0 C3 33 E0 C3 2D E0 0E 1B CD : 5D
E030 : 36 E3 C9 0E 1C CD 36 E3 23 4E CD 3A E3 2B 4E CD : 93
E040 : 3A E3 C9 0E 1E CD 36 E3 4E CD 3A E3 CD 6C E3 13 : 5F
E050 : 12 CD 6C E3 1B 12 C9 0E 1D CD 36 E3 4E CD 3A E3 : 6D
E060 : 13 1A 4F CD 3A E3 1B 1A 4F CD 3A E3 C9 7E 23 66 : A4
E070 : 6F 1A 08 13 1A 57 08 5F D5 0A 08 03 0A 57 08 5F : 2E
E080 : C1 E5 B7 ED 52 E1 C8 38 11 C5 CD B1 E0 4F CD CC : 99
E090 : E0 23 13 C1 0B 79 B0 20 F0 C9 09 2B EB 09 2B EB : 22
E0A0 : C5 CD B1 E0 4F CD CC E0 2B 1B C1 0B 79 B0 20 F0 : 36
E0B0 : C9 0E 0B CD 36 E3 4C CD 3A E3 4D CD 3A E3 0E 00 : 43
E0C0 : CD 3A E3 0E 01 CD 3A E3 CD 6C E3 C9 C5 0E 0C CD : 74
E0D0 : 36 E3 4A CD 3A E3 4B CD 3A E3 0E 00 CD 3A E3 0E : 88
E0E0 : 01 CD 3A E3 C1 CD 3A E3 C9 7E 23 66 6F 1A 13 08 : 0A
E0F0 : 1A 57 08 5F 0A 4F C5 0E 0C CD 36 E3 4C CD 3A E3 : 2C
-----------------------------------------------------------
 SUM : 61 28 46 23 C6 A3 6C 49 CB 71 21 7D AC 60 B8 9C : 4A
Complete
```

アミの部分がチェックサムの値です。

リスト 10-22 サブシステム用タイニーモニター

```
1000 '
1010 ' --- Monitor of sub-system ver 1.00 ---
1020 '
1030   CLEAR ,&HBFFF:DEFINT A-Z:DIM VALUE(12)
1040   WIDTH 80,25:CONSOLE 0,25,0,0
1050   MAXDRV=&HEC7D:MAXFLP=&HEF60:MAXMIN=&HEF62:MAXINT=&HEF62
1060   SGNBYT=&HEF63:DRVTBL=&HEF64
1070   IF PEEK(SGNBYT)=0 THEN "This utility can not run on PC-8031.1W":END
1080   BLOAD "edit.mac",&HE000
1100 '
1110 ' --- Display frame ---
1120 '
1130   LOCATE 0,0:PRINT ";"+STRING$(73,ASC("-"))+";"
1140   PRINT "; Monitor of sub_system ( 5'disk system )  ";
1150   PRINT "(Command mode)     (Edit mode)  ;"
1160   PRINT ";-"+STRING$(72,45)+";"
1170   PRINT ";-"+STRING$(72,32)+";"
1180   PRINT ";-"+STRING$(55,45)+";"+STRING$(16,45)+";"
1190   PRINT "; XX00 - 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ; ";
1200   PRINT "(CODE) - (ASC) ;"
1210   PRINT ";-"+STRING$(55,45)+";"+STRING$(16,45)+";"
1220   FOR I=0 TO 15
1230    PRINT ";"+SPACE$(56)+";"+SPACE$(16)+";"
1240   NEXT
1250   PRINT ";-"+STRING$(72,45)+";";
1260   CMDMODE=0:GOSUB *DSPCMD
1300 '
1310 ' --- Get command ---
1320 '
1330 *COMMANDMODE
1340   ON HELP GOSUB *CANCEL:HELP ON
```

```
1350   CMDMODE=0:GOSUB *DSPCMD:CODEMODE=2:GOSUB *DSPCODE
1360   LOCATE 0,3:PRINT ";-"+STRING$(72,32)+";"
1370   LOCATE 3,3
1380   A=ASC(INPUT$(1)):A=A AND &HDF:IF A<65 OR A>90 THEN GOTO 1380
1390   IF A=ASC("E") THEN PRINT "Edit "                  ;:GOTO *EDITMODE
1400   IF A=ASC("T") THEN PRINT "Transport  "            ;:GOTO *TRANSPORT
1410   IF A=ASC("D") THEN PRINT "Dump  "                 ;:GOTO *DUMP
1420   IF A=ASC("F") THEN PRINT "Full  "                 ;:GOTO *FULL
1430   IF A=ASC("G") THEN PRINT "Go  "                   ;:GOTO *GO
1440   IF A=ASC("M") THEN PRINT "Move  "                 ;:GOTO *MOVE
1450   IF A=ASC("R") THEN PRINT "Register "              ;:GOTO *REGISTER
1460   IF A=ASC("B") THEN PRINT "Set break point "       ;:GOTO *SETBREAK
1470   IF A=ASC("C") THEN PRINT "Continue  "             ;:GOTO *CONTINUE
1480   IF A=ASC("Q") THEN PRINT "Quit  "                 ;:HELP OFF:CLS:END
1490   GOTO *COMMANDMODE
1500 '
1510 ' --- Push help key ---
1520 '
1530 *CANCEL
1540   RETURN *COMMANDMODE
2000 '
2010 ' --- Transport data ---
2020 '
2030 *TRANSPORT
2040   TRANSM=&HE012:TRANSS=&HE015
2050   A=ASC(INPUT$(1)) AND &HDF:TYPE$=CHR$(A)
2060   IF A=ASC("M") THEN PRINT "(Main => Sub system) ";:GOTO 2090
2070   IF A=ASC("S") THEN PRINT "(Sub => Main system) ";:GOTO 2090
2080   GOTO 2050
2090   PRINT "Start:"; :GOSUB *GETADD:FA=VAL("&H"+ADD$):FA2!=ADD!
2100   PRINT "  End:"; :GOSUB *GETADD:EA=VAL("&H"+ADD$):EA2!=ADD!
2110   PRINT "  Dest:";:GOSUB *GETADD:DA=VAL("&H"+ADD$):DA2!=ADD!
2120   IF FA2! > EA2! THEN ER$="First add > End add":GOTO *TERROR
2130   SIZE2!=EA2!-FA2!+1:SIZE=VAL("&H"+HEX$(SIZE2!))
2140   IF TYPE$="M" THEN CALL TRANSM(FA,SIZE,DA):GOTO *COMMANDMODE
2150   IF DA2!+SIZE2!>=57345! THEN ER$="  Destroy program  ":GOTO *TERROR
2160   IF DA2!<49152! THEN ER$="  Destroy program  ":GOTO *TERROR
2170   CALL TRANSS(FA,SIZE,DA)
2180   GOTO *COMMANDMODE
2200 '
2210 ' --- Can not transport ---
2220 '
2230 *TERROR
2240   LOCATE 0,3:BEEP:PRINT ";- <  Abnormal termination.  ( ";ER$;" )  ";
2250   PRINT "Push return key.  >";:COLOR@ (53,3)-(70,3),6
2260   IF INKEY$<>CHR$(13) THEN 2260
2270   LOCATE 0,3:PRINT ";- Transport  "+STRING$(60,32)+";";:LOCATE 14,3
2280   GOTO *TRANSPORT
3000 '
3010 ' --- Full memory ---
3020 '
3030 *FULL
3040   PRINT " First add:";:GOSUB *GETADD:FA=VAL("&H"+ADD$):FA2!=ADD!
3050   PRINT " End add:";  :GOSUB *GETADD:EA=VAL("&H"+ADD$):EA2!=ADD!
3060   PRINT " Constant:"; :GOSUB *GETADD:CONSTANT=VAL("&H"+ADD$)
3070   IF FA2! > EA2! THEN ER$="First add > End add":GOTO *FERROR
3080   FULL=&HE018
3090   SIZE=VAL("&H"+HEX$(EA2!-FA2!+1))
3100   CALL FULL(FA,SIZE,CONSTANT)
3110   GOTO *COMMANDMODE
3200 '
3210 ' --- Can not full ---
3220 '
3230 *FERROR
3240   LOCATE 0,3:BEEP:PRINT ";- <  Abnormal termination.  ( ";ER$;" )  ";
3250   PRINT "Push return key.  >";:COLOR@ (53,3)-(70,3),6
3260   IF INKEY$<>CHR$(13) THEN 3260
3270   LOCATE 0,3:PRINT ";- Full  "+STRING$(65,32)+";";:LOCATE 9,3
3280   GOTO *FULL
4000 '
```

```
4010 ' --- Move data ---
4020 '
4030 *MOVE
4040   PRINT " First add:";        :GOSUB *GETADD:FA=VAL("&H"+ADD$):FA2!=ADD!
4050   PRINT " End add:";          :GOSUB *GETADD:EA=VAL("&H"+ADD$):EA2!=ADD!
4060   PRINT " Destination add:";:GOSUB *GETADD:DA=VAL("&H"+ADD$)
4070   IF FA2! > EA2! THEN ER$="First add > End add":GOTO *MERROR
4080   MOVE=&HE01B
4090   SIZE=VAL("&H"+HEX$(EA2!-FA2!+1))
4100   CALL MOVE(FA,SIZE,DA)
4110   GOTO *COMMANDMODE
4200 '
4210 ' --- Can not move ---
4220 '
4230 *MERROR
4240   LOCATE 0,3:BEEP:PRINT ";- <  Abnormal termination.  ( ";ER$;" )   ";
4250   PRINT "Push return key.  >";:COLOR@ (53,3)-(70,3),6
4260   IF INKEY$<>CHR$(13) THEN 4260
4270   LOCATE 0,3:PRINT ";- Move  "+STRING$(65,32)+";";:LOCATE 9,3
4280   GOTO *MOVE
5000 '
5010 ' --- Run user's program ---
5020 '
5030 *GO
5040   PRINT " Jump address:";:GOSUB *GETADD:JUMPADD=VAL("&H"+ADD$)
5050   GO=&HE01E
5060   CALL GO(JUMPADD)
5070   GOTO *COMMANDMODE
6000 '
6010 ' --- Run program when use break point ---
6020 '
6030 *CONTINUE
6040   GETREG=&HE024:PCNUM=11:CALL GETREG(PCNUM,VALUE)
6050   PRINT " (Now, program counter is ";RIGHT$("0000"+HEX$(VALUE),4);
6060   PRINT ")  --  Push return key.";:COLOR@ (49,3)-(66,3),6
6070   IF INKEY$<>CHR$(13) THEN 6070
6080   CONTBK=&HE02A
6090   CALL CONTBK
6100   GOTO *COMMANDMODE
6500 '
6510 ' --- Set break point ---
6520 '
6530 *SETBREAK
6540   PRINT " Break point:";:GOSUB *GETADD:BREAKPOINT=VAL("&H"+ADD$)
6550   SETBRK=&HE027
6560   CALL SETBRK(BREAKPOINT)
6570   GOTO *COMMANDMODE
7000 '
7010 ' --- Dump or set register ---
7020 '
7030 *REGISTER
7040   GETREG=&HE024:SETREG=&HE021
7050   A=ASC(INPUT$(1)) AND &HDF
7060   IF A=ASC("D") THEN *DUMPREG
7070   IF A=ASC("S") THEN *SETREG
7080   GOTO *REGISTER
7100 '
7110 ' --- Set register ---
7120 '
7130 *SETREG
7140   PRINT ", set.  Register name:";
7150   A$=""
7160   FOR I=1 TO 4
7170    A1$=INPUT$(1):IF A1$=" " OR A1$=CHR$(13) THEN 7230
7180    IF A1$="'" THEN 7210
7190    A1$=CHR$(ASC(A1$) AND &HDF)
7200    IF A1$<"A" OR A1$>"Z" THEN 7170
7210    PRINT A1$;:A$=A$+A1$
7220   NEXT
7230   RESTORE 7490
```

```
7240   FOR REGNUM=0 TO 12
7250    READ B$:IF A$=B$ THEN 7330
7260   NEXT REGNUM
7270   GOTO *RSERROR
7300 '
7310 ' --- change data ---
7320 '
7330   PRINT " (Now , ";
7340   CALL GETREG(REGNUM,VALUE)
7350   PRINT RIGHT$("0000"+HEX$(VALUE),4);") Value:";
7360   GOSUB *GETADD:VALUE=VAL("&H"+ADD$)
7370   CALL SETREG(REGNUM,VALUE)
7380   GOTO *COMMANDMODE
7400 '
7410 ' --- Register not found ---
7420 '
7430 *RSERROR
7440   LOCATE 0,3:BEEP:PRINT ";- <  ";A$;" register not found.";SPACE$(15)
7450   LOCATE 31,3:PRINT "Push return key.  >";:COLOR@ (30,3)-(47,3),6
7460   IF INKEY$<>CHR$(13) THEN 7460
7470   LOCATE 0,3:PRINT ";- Register "+SPACE$(45);:LOCATE 11,3
7480   GOTO *SETREG
7490   DATA AF,BC,DE,HL,AF',BC',DE',HL',IX,IY,I,PC,SP
7500 '
7510 ' --- Dump register ---
7520 '
7530 *DUMPREG
7540   PRINT ", dump"
7550   FOR REGNUM=0 TO 12
7560    CALL GETREG(REGNUM,VALUE(REGNUM))
7570   NEXT REGNUM
7580   LOCATE 0,7
7590   FOR I=0 TO 15
7600    PRINT ";"+SPACE$(56)+";"+SPACE$(16)+";"
7610   NEXT
7620   LOCATE 0,9:PRINT ";                    ----- Register dump -----"
7630   RESTORE 7770
7640   FOR I=0 TO 12
7650    IF (I MOD 5)=0 THEN PRINT:PRINT ";      ";
7660    READ A$:PRINT A$+":"+RIGHT$("0000"+HEX$(VALUE(I)),4)+"  ";
7670   NEXT
7680   PRINT:PRINT
7690   PRINT ";      ----- Sequence of flags in F register -----"
7700   PRINT
7710   PRINT ";               S   Z   X   H   X  P/V  N   C "
7720   PRINT ";              ";:F=VALUE(0)-INT(VALUE(0)/256)*256
7730   FOR I=7 TO 0 STEP -1
7740    IF (F AND (2^I))=0 THEN PRINT "  0 "; ELSE PRINT "  1 ";
7750   NEXT
7760   GOTO *COMMANDMODE
7770   DATA "AF ","BC ","DE ","HL ",AF',BC',DE',HL'
7780   DATA "IX ","IY ","I  ","PC ","SP "
8000 '
8010 ' --- Edit data ----
8020 '
8030 *EDITMODE
8040   PRINT "First:";:GOSUB *GETADD:KADD=VAL("&H"+ADD$)
8050   CMDMODE=1:GOSUB *DSPCMD:CODEMODE=0:GOSUB *DSPCODE
8060   LOCATE 0,7
8070   DSPMAC=&HE000:DATGET=&HE003:DATPUT=&HE006:DSPUP=&HE009:DSPDOWN=&HE00C
8080   KX=9:KY=7:KAX=58:KAY=7
8090   CHANGE=0:X=KX:Y=KY:AX=KAX:AY=KAY:HLFLAG=1
8100   CALL DSPMAC(KADD)
8110   ON HELP GOSUB *CHANGFLG
8120 *GETHEX
8130   LOCATE X,Y:HELP ON:A=ASC(INPUT$(1)):HELP OFF
8140 '
8150   IF HLFLAG=1 THEN LSIZE=2:RSIZE=1 ELSE LSIZE=1:RSIZE=2
8160 '
8170   IF A=13      THEN *COMMANDMODE
```

```
8180 '
8190   IF A=28       THEN X=X+RSIZE : HLFLAG=LSIZE MOD 2   ELSE 8230
8200    IF X=KX+48   THEN Y=Y+1     : X=KX : HLFLAG=1      ELSE 8230
8210     IF Y=KY+16 THEN Y=KY+15    : GOSUB *UPMAC
8220 '
8230   IF A=29       THEN X=X-LSIZE : HLFLAG=LSIZE MOD 2   ELSE 8270
8240    IF X=KX-2    THEN Y=Y-1     : X=KX+46 : HLFLAG=0   ELSE 8270
8250     IF Y=KY-1   THEN Y=KY      : GOSUB *DOWNMAC
8260 '
8270   IF A=30 THEN Y=Y-1 : IF Y=KY-1  THEN Y=KY : GOSUB *DOWNMAC
8280 '
8290   IF A=31 THEN Y=Y+1 : IF Y=KY+16 THEN Y=KY+15 : GOSUB *UPMAC
8300 '
8310   LOCATE X,Y
8320   IF A>=48 AND A<=57 THEN P=A-48:GOTO *SETHEX
8330   IF A>=65 AND A<=70 THEN P=A-55:GOTO *SETHEX
8340   IF A>=97 AND A<=102 THEN P=A-87:GOTO *SETHEX
8350   GOTO *GETHEX
8360 *SETHEX
8370   LOCATE X,Y:PRINT HEX$(P)
8380   IF HLFLAG=1 THEN OX=KX ELSE OX=KX+1
8390   OY=KY
8400   DUM=VAL("&H"+HEX$(KADD+(Y-OY)*16+(X-OX)¥3))
8410   CALL DATGET(DUM,M)
8420   IF HLFLAG=1 THEN M=M-INT(M/16)*16+P*16 ELSE M=INT(M/16)*16+P
8430   DUM=VAL("&H"+HEX$(KADD+(Y-OY)*16+(X-OX)¥3))
8435   CALL DATPUT(DUM,M)
8440 *SETASC
8450   IF (M < 32) OR (M >247)THEN M=ASC(".")
8460   LOCATE KAX+(X-OX)¥3,Y:PRINT CHR$(M)
8470 '
8480   X=X+RSIZE:HLFLAG=LSIZE MOD 2
8490   IF X=KX+48  THEN Y=Y+1    : X=KX : HLFLAG=1   ELSE 8520
8500    IF Y=KY+16 THEN Y=KY+15 : GOSUB *UPMAC      :REM right
8510 '
8520   LOCATE X,Y:GOTO *GETHEX
8530 *SET128
8540   M=ASC(MID$(B2$,(Y-OY)*16+(X-OX)¥3+1,1))
8550   IF HLFLAG=1 THEN M=M-INT(M/16)*16+P*16 ELSE M=INT(M/16)*16+P
8560   MID$(B2$,(Y-OY)*16+(X-OX)¥3+1,1)=CHR$(M)
8570   GOTO *SETASC
8580 *CHANGFLG
8590   HELP OFF
8600   IF CHANGE=1 THEN CHANGE=0:CODEMODE=0:GOSUB *DSPCODE:RETURN *GETHEX
8610   CHANGE=1:CODEMODE=1:GOSUB *DSPCODE:RETURN *GETASC
8620 *GETASC
8630   LOCATE AX,AY:HELP ON:A=ASC(INPUT$(1)):HELP OFF
8640 '
8650   IF A=13 THEN *COMMANDMODE
8660 '
8670   IF A=28         THEN AX=AX+1                          ELSE 8710
8680    IF AX=KAX+16   THEN AY=AY+1     : AX=KAX             ELSE 8710
8690     IF AY=KAY+16 THEN AY=KAY+15 : GOSUB *UPMAC
8700 '
8710   IF A=29         THEN AX=AX-1                           ELSE 8750
8720    IF AX=KAX-1    THEN AY=AY-1      : AX=KAX+15          ELSE 8750
8730     IF AY=KAY-1   THEN AY=KAY       : GOSUB *DOWNMAC
8740 '
8750   IF A=30 THEN AY=AY-1 : IF AY=KAY-1  THEN AY=KAY : GOSUB *DOWNMAC
8760 '
8770   IF A=31 THEN AY=AY+1 : IF AY=KAY+16 THEN AY=KAY+15 : GOSUB *UPMAC
8780 '
8790   LOCATE AX,AY:IF A>=28 AND A<=31 THEN *GETASC
8800   IF (A < 32) OR (A > 247) THEN A=ASC(".")
8810   PRINT CHR$(A)
8820   MX=(AX-KAX)*3+KX:MY=AY
8830   LOCATE MX,MY:PRINT RIGHT$("0"+HEX$(A),2)
8840   MX=MX+1
8850   DUM=VAL("&H"+HEX$(KADD+(MY-KY)*16+(MX-KX)¥3))
8860   CALL DATPUT(DUM,A)
```

```
8880 '
8890   AX=AX+1
8900   IF AX=KAX+16  THEN AY=AY+1   : AX=KAX         ELSE 8930
8910    IF AY=KAY+16 THEN AY=KAY+15 : GOSUB *UPMAC  :REM right
8920 '
8930   LOCATE AX,AY:GOTO *GETASC
9000 '
9010 ' --- Dump memory to printer ---
9020 '
9030 *DUMP
9040   RD256=&HE00F
9050   PRINT " First add:";:GOSUB *GETADD:FIRSTBLK=VAL("&H"+LEFT$(ADD$,2))
9060   PRINT " End add:";:GOSUB *GETADD:ENDBLK=VAL("&H"+LEFT$(ADD$,2))
9070   FOR BLKNUM=FIRSTBLK TO ENDBLK
9080    K2ADD=BLKNUM*256:NULLBUFF=VARPTR(#0)+9
9090    CALL RD256(K2ADD,NULLBUFF)
9100    FIELD #0 , 128 AS BLK1$ , 128 AS BLK2$
9110    LPRINT "       ===== Dump memory of sub-system (";
9120    LPRINT RIGHT$("0000"+HEX$(K2ADD),4);" - ";
9130    LPRINT RIGHT$("0000"+HEX$(K2ADD+255),4);")  ====="
9140    LPRINT
9150    LPRINT "          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F"
9160    R$=BLK1$:OFFSET=0:GOSUB *PRTDATA
9170    R$=BLK2$:OFFSET=1:GOSUB *PRTDATA
9180    IF (BLKNUM-FIRSTBLK+1) MOD 3 THEN LPRINT:LPRINT ELSE LPRINT CHR$(12);
9190   NEXT
9200   GOTO *COMMANDMODE
9210 *PRTDATA
9220   FOR J=0 TO 7
9230    LPRINT " ";RIGHT$("0000"+HEX$(K2ADD+(J+OFFSET*8)*16),4)+": ";
9240    U$=""
9250    FOR K=1 TO 16
9260     V$=MID$(R$,K+J*16,1):LPRINT RIGHT$("0"+HEX$(ASC(V$)),2)+" ";
9270     IF V$<" " OR V$>"秒" THEN V$="."
9280     U$=U$+V$
9290    NEXT K
9300    LPRINT "  ";U$
9310   NEXT J
9320   RETURN
9330 *DSPDATA
9340   FOR J=0 TO 7
9350    PRINT "; "+RIGHT$("0000"+HEX$(J+OFFSET*8),4)+" - ";
9360    U$=""
9370    FOR K=1 TO 16
9380     V$=MID$(R$,K+J*16,1):PRINT RIGHT$("0"+HEX$(ASC(V$)),2)+" ";
9390     IF V$<" " OR V$>"秒" THEN V$="."
9400     U$=U$+V$
9410    NEXT K
9420    PRINT ";";U$
9430   NEXT J
9440   RETURN
9450 *DSPCMD
9460   IF CMDMODE=0 THEN COLOR@ (42,1)-(57,1),4:COLOR@(60,1)-(72,1),0:RETURN
9470   COLOR@ (42,1)-(57,1),0:COLOR@(60,1)-(72,1),4:RETURN
9480 *DSPCODE
9490   IF CODEMODE=0 THEN COLOR@ (59,5)-(64,5),4:COLOR@(68,5)-(72,5),0:RETURN
9500   IF CODEMODE=1 THEN COLOR@ (59,5)-(64,5),0:COLOR@(68,5)-(72,5),4:RETURN
9510   COLOR@ (59,5)-(64,5),0:COLOR@(68,5)-(72,5),0:RETURN
9520 *UPMAC
9530   KADD=VAL("&H"+HEX$(KADD+16))
9540   CALL DSPUP(KADD)
9550   RETURN
9560 *DOWNMAC
9565   IF KADD<0 THEN KADD2!=KADD+65536! ELSE KADD2!=KADD
9570   KADD=VAL("&H"+HEX$(KADD2!-16))
9580   CALL DSPDOWN(KADD)
9590   RETURN
9600 *GETADD
9610   ADD$=""
9620   A=ASC(INPUT$(1))
```

```
9630    IF A=13 OR A=32 THEN *RETADD
9640    IF A=127 AND ADD$<>"" THEN *DELADD
9650    IF LEN(ADD$)=4 THEN 9620
9660    IF A>&H2F AND A<&H3A THEN 9680
9670    A=A AND &HDF:IF A<&H41 OR A>&H46 THEN 9620
9680    PRINT CHR$(A);:ADD$=ADD$+CHR$(A)
9690    GOTO 9620
9700 *DELADD
9710    ADD$=LEFT$(ADD$,LEN(ADD$)-1)
9720    PRINT CHR$(29)+" "+CHR$(29);
9730    GOTO 9620
9800 *RETADD
9810    PRINT " ";
9820    ADD$=RIGHT$("0000"+ADD$,4)
9830    ADD!=VAL("&H"+ADD$):IF ADD!<0 THEN ADD!=ADD!+65536!
9840    RETURN

E000 : C3 98 E2 C3 91 E1 C3 B2 E1 C3 D1 E1 C3 1F E2 C3 : C4
E010 : 6C E2 C3 29 E1 C3 5D E1 C3 E9 E0 C3 6D E0 C3 17 : 92
E020 : E1 C3 57 E0 C3 43 E0 C3 33 E0 C3 2D E0 0E 1B CD : 5D
E030 : 36 E3 C9 0E 1C CD 36 E3 23 4E CD 3A E3 2B 4E CD : 93
E040 : 3A E3 C9 0E 1E CD 36 E3 4E CD 3A E3 CD 6C E3 13 : 5F
E050 : 12 CD 6C E3 1B 12 C9 0E 1D CD 36 E3 4E CD 3A E3 : 6D
E060 : 13 1A 4F CD 3A E3 1B 1A 4F CD 3A E3 C9 7E 23 66 : A4
E070 : 6F 1A 08 13 1A 57 08 5F D5 0A 08 03 0A 57 08 5F : 2E
E080 : C1 E5 B7 ED 52 E1 C8 38 11 C5 CD B1 E0 4F CD CC : 99
E090 : E0 23 13 C1 0B 79 B0 20 F0 C9 09 2B EB 09 2B EB : 22
E0A0 : C5 CD B1 E0 4F CD CC E0 2B 1B C1 0B 79 B0 20 F0 : 36
E0B0 : C9 0E 0B CD 36 E3 4C CD 3A E3 4D CD 3A E3 0E 00 : 43
E0C0 : CD 3A E3 0E 01 CD 3A E3 CD 6C E3 C9 C5 0E 0C CD : 74
E0D0 : 36 E3 4A CD 3A E3 4B CD 3A E3 0E 00 CD 3A E3 0E : 88
E0E0 : 01 CD 3A E3 C1 CD 3A E3 C9 7E 23 66 6F 1A 13 08 : 0A
E0F0 : 1A 57 08 5F 0A 4F C5 0E 0C CD 36 E3 4C CD 3A E3 : 2C
---------------------------------------------------------
 SUM : 61 28 46 23 C6 A3 6C 49 CB 71 21 7D AC 60 B8 9C : 4A

E100 : 4D CD 3A E3 4A CD 3A E3 4B CD 3A E3 C1 CD 3A E3 : 4B
E110 : 23 1B 7B B2 20 F7 C9 7E 23 66 6F 0E 0D CD 36 E3 : C2
E120 : 4C CD 3A E3 4D CD 3A E3 C9 7E 23 66 6F E5 1A 13 : BE
E130 : 08 1A 57 08 5F 0A 03 08 0A 67 08 6F 0E 0C CD 36 : FA
E140 : E3 4C CD 3A E3 4D CD 3A E3 4A CD 3A E3 4B CD 3A : D6
E150 : E3 E1 4E CD 3A E3 23 1B 7B B2 20 F6 C9 7E 23 66 : 4D
E160 : 6F 1A 13 08 1A 57 08 5F 0A 03 08 0A 47 08 4F C5 : FE
E170 : 0E 0B CD 36 E3 4C CD 3A E3 4D CD 3A E3 4A CD 3A : BD
E180 : E3 4B CD 3A E3 E1 CD 6C E3 77 23 1B 7B B2 20 F6 : 0D
E190 : C9 0E 0B CD 36 E3 23 4E CD 3A E3 2B 4E CD 3A E3 : 86
E1A0 : 0E 00 CD 3A E3 0E 01 CD 3A E3 CD 6C E3 12 13 AF : E1
E1B0 : 12 C9 0E 0C CD 36 E3 23 4E CD 3A E3 2B 4E CD 3A : B6
E1C0 : E3 0E 00 CD 3A E3 0E 01 CD 3A E3 1A 4F CD 3A E3 : 27
E1D0 : C9 0E 0B CD 36 E3 5E 23 56 21 F0 00 19 4C CD 3A : 1C
E1E0 : E3 4D CD 3A E3 0E 00 CD 3A E3 0E 10 CD 3A E3 11 : 2B
E1F0 : 00 E5 06 10 CD 6C E3 12 13 10 F9 E5 2E 09 CD 1E : 4C
---------------------------------------------------------
 SUM : 62 91 D2 F6 19 B6 28 E7 34 13 7D DE 5B E1 54 BC : 87

E200 : 43 D5 2E 08 CD 1E 43 E1 01 08 07 ED B0 E1 11 00 : FC
E210 : E5 3E 17 32 86 EF 3E 01 32 87 EF CD BB E2 C9 0E : 09
E220 : 0B CD 36 E3 7E 23 66 6F 4C CD 3A E3 4D CD 3A E3 : D4
E230 : 0E 00 CD 3A E3 0E 10 CD 3A E3 11 00 E5 06 10 CD : D9
E240 : 6C E3 12 13 10 F9 E5 2E 17 CD 1E 43 1B D5 2E 18 : 0B
E250 : CD 1E 43 1B E1 01 08 07 ED B8 E1 11 00 E5 3E 08 : FC
E260 : 32 86 EF 3E 01 32 87 EF CD BB E2 C9 7E 23 66 6F : 37
E270 : 1A 08 13 1A 57 08 5F 0E 0B CD 36 E3 4C CD 3A E3 : 42
E280 : 4D CD 3A E3 0E 01 CD 3A E3 0E 00 CD 3A E3 06 00 : 2E
E290 : CD 6C E3 12 13 10 F9 C9 7E 23 66 6F 11 00 E5 CD : 4C
E2A0 : 77 E2 11 00 E5 3E 08 32 86 EF 3E 01 32 87 EF 0E : 31
E2B0 : 10 CD BB E2 0D C4 F9 E2 20 F7 C9 3E 3B DF 3E 20 : BC
E2C0 : DF CD 02 E3 3E 20 DF 3E 2D DF 3E 20 DF D5 06 10 : 40
E2D0 : 1A CD 07 E3 3E 20 DF 13 10 F6 D1 3E 3B DF 06 10 : 66
```

```
E2E0 : 1A FE F8 30 04 FE 20 30 02 3E 2E DF 13 10 F1 3E : 31
E2F0 : 3B DF D5 11 10 00 19 D1 C9 F5 3E 0D DF 3E 0A DF : 09
-----------------------------------------------------------
 SUM : B5 CE 5E BB A0 C3 88 B9 A4 6B 40 62 46 8B 4F 68 : 79

E300 : F1 C9 7C CD 07 E3 7D D5 F5 CD 13 E3 7A DF 7B DF : AA
E310 : F1 D1 C9 F5 CD 22 E3 5F F1 07 07 07 07 CD 22 E3 : 90
E320 : 57 C9 E6 0F CD 28 E3 C9 FE 0A 30 03 F6 30 C9 FE : DE
E330 : 10 3F D8 C6 37 C9 3E 0F D3 FF CD 9A E3 DB FE CB : FA
E340 : 4F 20 05 CD A7 E3 18 F5 3E 0E D3 FF 79 D3 FD 3E : 7D
E350 : 09 D3 FF DB FE CB 57 20 05 CD A7 E3 18 F5 3E 08 : A5
E360 : D3 FF DB FE E6 04 C8 CD A7 E3 18 F6 CD 9A E3 3E : 4A
E370 : 0B D3 FF DB FE CB 47 20 05 CD A7 E3 18 F5 3E 0A : 99
E380 : D3 FF DB FC 4F 3E 0D D3 FF DB FE CB 47 28 05 CD : FA
E390 : A7 E3 18 F5 3E 0C D3 FF 79 C9 AF 32 C6 E3 32 C5 : 76
E3A0 : E3 3E 00 32 C4 E3 C9 3A C6 E3 3D 32 C6 E3 C0 3A : B8
E3B0 : C5 E3 3D 32 C5 E3 C0 3A C4 E3 3D 32 C4 E3 C0 F1 : 27
E3C0 : F1 C3 2B A7 00 00 FE 00 00 00 00 00 00 00 00 00 : 84
E3D0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
E3E0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
E3F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : 00
-----------------------------------------------------------
 SUM : 92 2D 3C 14 77 83 66 54 A8 D2 77 A3 67 DF 77 D6 : EA
```

10-4-2　片面、両面切り換え

NECのシステムディスクには、片面ディスクシステムと両面ディスクシステムの２つがあります。PC-8801のシステムディスクには“SStoDS”,“DStoSS”の2つのユーティリティがあり、データの交換ができたのですが、PC-8801mkⅡには添付されていません。

そこでディスクシステムを自由に片面、両面に切り換えるプログラムを載せておきます。

まずRunして下さい。ISETコマンドで設定ができるようになります。

①ISET⏎

現在のドライブの状態を表示します。

②ISET＜ドライブ番号＞、｜S｜ ⏎
｜D｜

ドライブ番号はfilesかLOADコマンドで使っているドライブ番号と同じ値を使い、５インチディスクユニットを指定します。そして片面システムにするならSを、両面システムならDを指定します。

ISET　1，S⏎……ドライブ１を片面にする。(片面のディスクを使用できます)

ISET　2，D⏎……ドライブ２を両面にする。

③IRESET⏎

すべてのドライブを両面モードに戻します。

≪注意≫

ストップリセットを行なったときは、必ずIRESETを実行して下さい。なお、このコマンドの原理は「15-15ドライブ２を片面として使う」に説明があります。

リスト 10-23 片面、両面切り換えプログラム

```
100 ' ┌──────────────────────────────────────────────────────────────────────┐
110 ' │ Disk drive mode change command for 5-2D intelligent drive unit       │
120 ' │                                                                      │
130 ' │    Last version  84/06/16              written by M.Kurata           │
140 ' │    Copyright 1984 (C) by SYSTEM SOFT                                 │
150 ' └──────────────────────────────────────────────────────────────────────┘
160 '
170  WIDTH 80,25:CONSOLE 0,25,,0
180 '
190  PRINT "--- Install 'ISET','IRESET' statements.---"
200  PRINT
210   GOSUB *STORE
220   GOSUB *LINK
230  PRINT "  Installation complete."
240  PRINT
250  PRINT "     ISET                   : Print drive status."
260  PRINT "     ISET   <Drive#>,|S|    : Set drive status."
270  PRINT "                     |D|"
280  PRINT "     IRESET                 : Set drives mode into double surface."
290  PRINT
300  PRINT "  This module uses 226 bytes (0F21EH-0F2FFH)."
310  PRINT
320 '
330 *EXIT
340  CLEAR:NEW:END
350 '------------------------------------------------------------------------
360 *LINK
370 '
380  MAXINT=PEEK(&HEF62):SGNBYT=PEEK(&HEF63)
390  MAXFLP=PEEK(&HEF60):MAXMIN=PEEK(&HEF61)
400 '
410  IF MAXINT<>0 AND SGNBYT<>0 THEN 450
420     PRINT "   Your drive is not sutable.  Not install.":BEEP
430    RETURN *EXIT
440 '
450  POKE &HF21E,MAXFLP+MAXMIN+1
460  POKE &HF21F,MAXINT
470  POKE &HEEAA,&HC3:POKE &HEEAB,&H24:POKE &HEEAC,&HF2        ' ISET
480  POKE &HEEAD,&HC3:POKE &HEEAE,&HC0:POKE &HEEAF,&HF2        ' IRESET
490 RETURN
500 '------------------------------------------------------------------------
510 *STORE
520 '
530  FOR I=&HF220 TO &HF2FF
540    READ W$:POKE I,VAL("&H"+W$)
550  NEXT
560 RETURN
570 '
580 ' Machine data
590 '
600 DATA 01,02,04,08,28,64,CD,A3,18,08,7B,E5,21,1E,F2,96
610 DATA DA,E2,F2,23,BE,D2,E2,F2,21,20,F2,85,6F,7E,E1,F5
620 DATA CF,2C,F5,D7,20,09,F1,1E,03,D6,44,28,0B,D6,0F,C2
630 DATA 93,03,F1,2F,F5,1D,3E,F0,C6,B0,32,78,F2,F1,E5,08
640 DATA 21,63,EF,85,6F,73,08,47,21,DD,F2,E5,E5,E5,E5,E5
650 DATA 3E,18,CD,C9,37,CD,47,38,00,47,3E,17,CD,C9,37,78
660 DATA CD,D2,37,E1,E1,E1,E1,E1,E1,C9,E5,21,E7,F2,CD,50
670 DATA 55,21,1E,F2,11,63,EF,7E,83,5F,3E,30,86,4F,23,46
680 DATA C5,D5,79,32,FA,F2,1A,FE,02,3E,53,28,02,D6,0F,32
690 DATA FC,F2,21,F2,F2,CD,50,55,D1,13,C1,0C,10,E2,E1,C9
700 DATA 20,8D,E5,21,1E,F2,11,63,EF,7E,83,5F,23,46,3E,03
710 DATA 12,13,10,FC,3E,B0,32,78,F2,3E,0F,18,8A,1E,40,C3
720 DATA B3,03,1E,46,C3,B3,03,53,74,61,74,75,73,3A,2D,0D
730 DATA 0A,00,20,20,44,72,69,76,65,20,31,2D,44,0D,0A,00
```

10-4-3　インタリーブフォーマット

第9章の8インチのインタリーブフォーマットプログラムの5インチ版です。Runして、指定されたドライブにディスクを入れ、"y" を押せば、インタリーブファクター13で物理的フォーマットを行ないます。DISK-BASICの立ち上げなどはこうすれば速くなりますが、例えばASCII社のDUAD-PC88Dでは逆に遅くなります。これはディスクを読むのに必要な時間の違いから来るもので、それ専用にファクターを決めてやれば解決できます。

インタリーブのファクターを変える場合は、940行のデータを書き変えて下さい。このデータの順番にセクタ番号が設定されます。

実行例

```
------ Interleave format.88 ---------

Format a disk on drive 2 . Sure(y/n) ? y

Complete

Ok
```

リスト 10-24 インタリーブフォーマット

```
100 '
110 ' ------ Interleave formatting. 5inch floppy version -------
120 '
130   WIDTH 80,25:CONSOLE 0,25,0,1:CLS
140   MAXFLP=&HEF60:MAXMIN=&HEF61:MAXINT=&HEF62
150   PA=&HFC:PB=&HFD:PC=&HFE:CM=&HFF:RESTORE *MACDATA
160   LOCATE 20,13:PRINT "Now , send data to sub_system."
170   A=12:GOSUB *SNDCOM
180   FOR I=0 TO 191
190    IF (I MOD 16)=0 THEN LOCATE 50,13:PRINT 12-(I ¥ 16);" ";
200    READ A$
210    A=VAL("&H"+A$):GOSUB *SNDPAR
220   NEXT
230   CLS
240 '
250 *MAIN
260 '
270    PRINT "------ Interleave format.88 ---------":PRINT
280    DRIVE=PEEK(MAXFLP)+PEEK(MAXMIN)+PEEK(MAXINT)
290    PRINT "Format a disk on drive";DRIVE;:INPUT ". Sure(y/n) ";DUM$:PRINT
300    IF DUM$<>"Y" AND DUM$<>"y" THEN GOTO 290
310 '
320 ' ----- Send drive and interleave factor to sub_system -----
330 '
340    A=12:GOSUB *SNDCOM:A=&H70:GOSUB *SNDPAR:A=&H3:GOSUB *SNDPAR
350    A=0:GOSUB *SNDPAR:A=17:GOSUB *SNDPAR
360    A=PEEK(MAXINT)-1:GOSUB *SNDPAR : REM Send # of formatting drive
370    RESTORE *FACTOR
380    FOR I=1 TO 16
390     READ A:GOSUB *SNDPAR : REM Send interleave factor (# of sector)
400    NEXT
410 '
420 ' -- Set program counter of sub_system to 7000H (Run intreleave program) --
430 '
440    A=13:GOSUB *SNDCOM:A=&H70:GOSUB *SNDPAR:A=0:GOSUB *SNDPAR
450 '
460 ' ------ Get status flag. If bit0 is 1 , then failed to format ------
470 '
480    A=6:GOSUB *SNDCOM:GOSUB *REVPAR:FLAG=A
490    IF FLAG=&H80 THEN PRINT "Complete":PRINT:END
500    PRINT "Disk I/O error":PRINT:BEEP:BEEP
510    END
```

```
900 '
910 ' ------ Table of interleave factor (Interleave factor = 13) --------
920 '
930 *FACTOR
940     DATA 1,14,11,8,5,2,15,12,9,6,3,16,13,10,7,4
950 '
960 '
970 '
1000 *SNDCOM
1010    OUT CM,&HF
1100 *SNDPAR
1110    IF (INP(PC) AND 2)=0 THEN 1110
1120    OUT CM,&HE
1130    OUT PB,A
1140    OUT CM,9
1150    IF (INP(PC) AND 4)=0 THEN 1150
1160    OUT CM,8
1170    IF (INP(PC) AND 4)=1 THEN 1170
1180    RETURN
1200 *REVPAR
1210    OUT CM,&HB
1220    IF (INP(PC) AND 1)=0 THEN 1220
1230    OUT CM,&HA
1240    A=INP(PA)
1250    OUT CM,&HD
1260    IF (INP(PC) AND 1)=1 THEN 1260
1270    OUT CM,&HC
1280    RETURN
10000 *MACDATA
10010 DATA 70,00,00,c0  :REM disk ram 7000h - 70BFh
10020 DATA C3,15,70,01,01,02,03,04,05,06,07,08,09,0A,0B,0C
10030 DATA 0D,0E,0F,10,00,3A,03,70,4F,CD,96,01,38,0F,16,00
10040 DATA CD,38,70,38,08,14,CD,1B,04,FE,28,38,F3,3E,00,17
10050 DATA F6,80,32,14,7F,C3,C1,00,CD,AA,01,D8,3E,05,32,0A
10060 DATA 7F,3E,4D,CD,A4,02,E5,C5,D5,CD,0E,04,E7,CD,2F,04
10070 DATA 47,CD,1B,04,57,3E,01,E7,3E,10,E7,3E,33,E7,3E,FF
10080 DATA E7,21,04,70,5E,FB,76,DB,FA,E6,20,28,2B,7A,D3,FB
10090 DATA FB,76,DB,FA,E6,20,28,20,78,D3,FB,FB,76,DB,FA,E6
10100 DATA 20,28,15,7B,D3,FB,FB,76,DB,FA,E6,20,28,0A,3E,01
10110 DATA D3,FB,23,7E,5F,B7,20,CD,FB,76,F3,D1,C1,E1,CD,63
10120 DATA 02,D0,3A,0E,7F,E6,02,37,C0,3A,0A,7F,3D,C2,3E,70
10130 DATA 37,C9,00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

10-4-4　MS-DOSフォーマットのディスクのリード・ライト

このプログラムはMS-DOSフォーマット(512バイト／セクタ)のディスクの、セクタ単位のリード・ライトを行ないます。

ユニット番号、シリンダ番号、ヘッド番号、セクタ番号を指定すると、リードなら、データをメインシステムのE000H～E1FFHに読み込んできます。また、ライトなら、メインシステムのE000H～E1FFHのデータを書き込みます。

実行例

```
+++++ Read / Write disk of MS-DOS (Ver 1.25 : Ver 2.0) +++++

Read or Write (R/W) ? r

Drive unit number (0 or 1) ? 1

Cylinder number (0 - 39) ? 0

Head (surface) number (0 or 1) ? 0

Sector number (1 - 9) ? 1

Recieve data (E000H - E1FFH) from sub-system.

Complete
```

リスト 10-25 MS-DOSフォーマットディスクのリード､ライト

```
100 REM  read and write disk of MS-DOS
110   CLEAR ,&HDFFF:WIDTH 80,25:CONSOLE 0,25,0,1:CLS
120   BUFF=&HE000:PA=&HFC:PB=&HFD:PC=&HFE:CM=&HFF:RESTORE *MACDATA
130   LOCATE 20,13:PRINT "Now , send data to sub-system."
140   A=12:GOSUB *SNDCOM
150   FOR I=0 TO 515
160    IF (I MOD 16)=0 THEN LOCATE 50,13:PRINT 32-(I ¥ 16);" ";
170    READ A$
180    A=VAL("&H"+A$):GOSUB *SNDPAR
190   NEXT
200   CLS
300 *MENU
310   PRINT
320   PRINT "+++++ Read / Write disk of MS-DOS (Ver 1.25 : Ver 2.0) +++++"
330   PRINT:INPUT "Read or Write (R/W) ";DUM$
340   IF DUM$="r" OR DUM$="R" THEN GOSUB *READMS:GOTO *MENU
350   IF DUM$="w" OR DUM$="W" THEN GOSUB *WRITEMS:GOTO *MENU
360   GOTO 330
400 *GETINFO
410   PRINT:INPUT "Drive unit number (0 or 1) ";DRIVE
420   IF DRIVE<0 OR DRIVE>1  THEN 410
430   PRINT:INPUT "Cylinder number (0 - 39) ";CYLINDER
440   IF CYLINDER<0 OR CYLINDER>39  THEN 430
450   PRINT:INPUT "Head (surface) number (0 or 1) ";HEAD
460   IF HEAD<0 OR HEAD>1 THEN 450
470   PRINT:INPUT "Sector number (1 - 9) ";SECTOR
480   IF SECTOR<1 OR SECTOR>9 THEN 470
490   RETURN
500 *SNDINFO
510   A=DRIVE:GOSUB *SNDPAR
520   A=CYLINDER:GOSUB *SNDPAR
530   A=HEAD:GOSUB *SNDPAR
540   A=SECTOR:GOSUB *SNDPAR
550   RETURN
600 *READMS
610   GOSUB *GETINFO
620   A=13:GOSUB *SNDCOM
```

```
630    A=&H70:GOSUB *SNDPAR
640    A=0:GOSUB *SNDPAR
650    GOSUB *SNDINFO
660    GOSUB *REVPAR:IF A=&H81 THEN PRINT "Disk I/O error":END
670    PRINT:PRINT "Recieve data (E000H - E1FFH) from sub-system.";
680    X=POS(0):Y=CSRLIN
690    FOR I=0 TO 511
700     IF (I MOD 16)=0 THEN LOCATE X,Y:PRINT 31-(I ¥ 16);" ";
710     GOSUB *REVPAR:POKE BUFF+I,A
720    NEXT
730    LOCATE X,Y:PRINT "  "
740    PRINT:PRINT "Complete":RETURN
800 *WRITEMS
810    GOSUB *GETINFO
820    A=13:GOSUB *SNDCOM
830    A=&H70:GOSUB *SNDPAR
840    A=3:GOSUB *SNDPAR
850    GOSUB *SNDINFO
860    PRINT:PRINT "Send data (E000H - E1FFH) to sub-system.";
870    X=POS(0):Y=CSRLIN
880    FOR I=0 TO 511
890     IF (I MOD 16)=0 THEN LOCATE X,Y:PRINT 31-(I ¥ 16);" ";
900     A=PEEK(BUFF+I):GOSUB *SNDPAR
910    NEXT
920    LOCATE X,Y:PRINT "  ":PRINT
930    GOSUB *REVPAR:IF A=&H81 THEN PRINT "Disk I/O error":END
940    PRINT "Complete":RETURN
1000 *SNDCOM
1010    OUT CM,&HF
1100 *SNDPAR
1110    IF (INP(PC) AND 2)=0 THEN 1110
1120    OUT CM,&HE
1130    OUT PB,A
1140    OUT CM,9
1150    IF (INP(PC) AND 4)=0 THEN 1150
1160    OUT CM,8
1170    IF (INP(PC) AND 4)=1 THEN 1170
1180    RETURN
1200 *REVPAR
1210    OUT CM,&HB
1220    IF (INP(PC) AND 1)=0 THEN 1220
1230    OUT CM,&HA
1240    A=INP(PA)
1250    OUT CM,&HD
1260    IF (INP(PC) AND 1)=1 THEN 1260
1270    OUT CM,&HC
1280    RETURN
10000 *MACDATA
10010 DATA 70,00,02,00  :REM disk ram 7000h - 71FFh
10020 DATA C3,12,70,C3,33,70,00,40,00,00,00,00,00,00,00,00
10030 DATA 00,01,CD,D6,70,CD,F4,70,CD,59,70,DA,53,70,3E,80
10040 DATA DF,2A,06,70,ED,4B,08,70,7E,DF,23,0B,79,B0,20,F8
10050 DATA C3,C1,00,CD,D6,70,CD,F4,70,2A,06,70,ED,4B,08,70
10060 DATA D7,77,23,0B,79,B0,20,F8,CD,94,70,38,06,3E,80,DF
10070 DATA C3,C1,00,3E,81,DF,C3,C1,00,CD,0F,71,D8,CD,9D,71
10080 DATA 3E,09,32,0A,7F,3E,46,CD,A4,02,CD,5A,71,2A,06,70
10090 DATA ED,4B,08,70,FB,76,DB,FA,E6,20,28,09,DB,FB,77,23
10100 DATA 0B,79,B0,20,EF,CD,77,71,D0,CD,B0,71,21,0A,7F,35
10110 DATA 20,D3,37,C9,CD,0F,71,D8,CD,9D,71,3E,09,32,0A,7F
10120 DATA 3E,45,CD,A4,02,CD,5A,71,2A,06,70,ED,4B,08,70,FB
10130 DATA 76,DB,FA,E6,20,28,09,7E,D3,FB,23,0B,79,B0,20,EF
10140 DATA CD,77,71,D0,3A,0E,7F,E6,02,37,C0,CD,B0,71,21,0A
10150 DATA 7F,35,20,CC,37,C9,D7,32,0A,70,57,D7,32,0B,70,32
10160 DATA 0E,70,D7,32,0C,70,32,0F,70,07,07,B2,32,0D,70,D7
10170 DATA 32,10,70,C9,21,DA,71,16,00,3E,02,07,5F,19,5E,23
10180 DATA 56,21,00,00,3A,11,70,47,19,10,FD,22,08,70,C9,3A
10190 DATA 0A,70,FE,04,3F,D8,3A,0B,70,FE,28,3F,D8,CD,CE,71
10200 DATA 3E,0F,CD,A4,02,3A,0A,70,E7,3A,0B,70,E7,FB,76,F3
10210 DATA 3E,08,E7,CD,9A,02,E6,DF,21,0A,70,AE,20,0B,CD,9A
```

```
10220 DATA 02,21,0B,70,BE,20,05,B7,C9,CD,9A,02,3E,07,CD,A4
10230 DATA 02,3A,0A,70,E7,FB,76,F3,37,C9,3A,0D,70,E7,3A,0E
10240 DATA 70,E7,3A,0F,70,E7,3A,10,70,E7,3E,02,E7,3E,09,E7
10250 DATA 3E,1B,E7,3E,FF,E7,C9,DB,F8,FB,76,F3,3A,0D,70,4F
10260 DATA 21,0D,7F,E5,06,07,CD,9A,02,77,23,10,F9,E1,3E,DF
10270 DATA A6,A9,23,B6,4F,23,3E,73,A6,B1,C8,37,C9,3A,0A,70
10280 DATA 21,00,7F,87,85,6F,5E,23,56,1A,D3,F7,32,0B,7F,C9
10290 DATA 3A,0A,70,21,00,7F,87,85,6F,5E,23,56,13,3E,F9,BB
10300 DATA 20,03,11,F0,07,1A,D3,F7,32,0B,7F,2B,73,C9,3A,0B
10310 DATA 70,FE,1E,3E,0F,30,02,3E,07,D3,F8,C9,00,01,00,02
10320 DATA 00,04,00,08,00,10,00,20,42,42,2A,17,5E,9C,44,16
10330 DATA 01,00,18,1A,5E,3E,3E,4A,0A,2E,A7,D9,EF,00,00,E7
```

第11章　プリンタ

PC-8801mkⅡのプリンタインタフェースは、セントロニクスインタフェースに準拠しています。これはアメリカのセントロニクス社のプリンタに用いられた８ビットのパラレルインタフェースなのですが、現在では多くの機種に採用され、標準的なプリンタインタフェースとなってています。

11-1　画面コピー

N88-BASICでは、COPY文とCOPYキーにより、画面上に表示されている文字や図形をプリンタに出力する機能をもっています。しかしプリンタに出力される文字のフォントは画面とは異なりますし、グラフィックも縦横の比が正しくなく(円が縦長になる)、カラーも無視されるなど、いくつかの欠点をもっています。ここではこれらを改良したプログラムを紹介します。

11-1-1　テキスト画面のコピー

次のプログラムはテキスト画面を画面と同じ文字フォントでコピーします。もちろん画面上で見えない文字(シークレットやCOLOR０)はプリントされませんし、リバース文字は反転して出力されます。(ただし、20行モードでのリバースでは行間があいてしまいます。)

このプログラムはPC-8821／8822用ですが、410行の最初のデータ(０E)を次のように変更すると、他のプリンタにも対応できます。

06　…　PC-8023(C)

0E　…　PC-8821／8822，NM-9100／9200，NK-3618

16　…　PC-PR201

1E　…　NM-9300／9400(S)

リスト　11-1

```
100 '
110 '   text copy
120 '
130 DEFINT A-Z
140 RESTORE *MAIN
150 FOR I=&H7000 TO &H7245
160   READ D$ : POKE I,VAL("&H"+D$)
170 NEXT
180 '
190 RESTORE *MOVM
200 FOR I=&HF320 TO &HF338
210   READ D$ : POKE I,VAL("&H"+D$)
220 NEXT
230 DEF USR=&HF320 : I=USR(0)
240 '
```

```
250 RESTORE *CALLER
260 FOR I=&HF320 TO &HF32F
270   READ D$ : POKE I,VAL("&H"+D$)
280 NEXT
290 '
300 END
310 '
320 *CALLER
330 DATA E5,DB,70,F5,3E,80,D3,70,CD,00,80,F1,D3,70,E1,C9
340 '
350 *MOVM
360 DATA 3A,C2,E6,F5,F6.02,F3,D3,31,21,00,70,11,00,80,01
370 DATA 00,03,ED,B0,F1,D3,31,FB,C9
380 '
390 *MAIN
400 ' 8000-807F
410 DATA 0E,00,C3,15,80,00,00,00,00,00,00,00,00,00,00,00
420 DATA 00,00,00,00,00,D3,EB,CD,69,80,2E,01,CD,C2,35,38
430 DATA 25,E5,CD,E7,80,E1,26,01,E5,CD,9D,42,CD,52,44,CD
440 DATA 4A,81,E1,24,3A,89,EF,BC,30,EE,E5,CD,3F,81,E1,2C
450 DATA 3A,88,EF,BD,30,D6,06,4E,3A,00,80,FE,06,28,02,06
460 DATA 48,3E,1B,CD,2D,82,78,CD,2D,82,21,61,80,CD,24,82
470 DATA C9,0F,1B,41,1B,5D,0D,0A,00,06,0F,3A,89,EF,FE,32
480 DATA 30,02,06,0E,78,CD,2D,82,3A,00,80,D6,06,6F,0F,85
490 ' 8080-80FF
500 DATA 6F,26,00,11,B7,80,19,5E,3A,88,EF,FE,15,30,03,23
510 DATA 5E,2B,D5,23,23,CD,24,82,D1,16,00,21,A3,80,19,CD
520 DATA 24,82,C9,1B,54,31,32,00,1B,54,31,35,00,1B,54,31
530 DATA 36,00,1B,54,32,30,00,0A,0F,1B,51,1B,5B,00,00,00
540 DATA 00,00,00,0A,0F,1B,48,1B,3E,00,00,00,00,00,00,00
550 DATA 05,1B,48,00,00,00,00,00,00,00,00,0A,0F,1B,51,1A
560 DATA 41,1A,46,1B,3E,00,00,3A,00,80,6F,26,00,11,F9,80
570 DATA 3A,89,EF,FE,32,30,03,11,19,81,19,CD,24,82,C9,1B
580 ' 8100-817F
590 DATA 53,30,36,34,30,00,00,1B,49,30,36,34,30,00,00,1B
600 DATA 49,30,36,34,30,00,00,1B,4A,30,36,34,30,00,00,1B
610 DATA 53,30,33,32,30,00,00,1B;49,30,33,32,30,00,00,1B
620 DATA 49,30,33,32,30,00,00,1B,4A,30,33,32,30,00,00,3E
630 DATA 0D,CD,2D,82,3E,0A,CD,2D,82,C9,C5,CD,C1,81,C1,CB
640 DATA 59,20,0C,CB,51,C4,AA,81,CB,41,C4,B6,81,18,06,79
650 DATA E6,E0,CC,B6,81,3A,00,80,1E,00,D6,06,28,07,1C,D6
660 DATA 11,38,02,1E,03,21,05,80,16,08,4E,3E,80,06,08,CB
670 ' 8180-81FF
680 DATA 09,1F,DC,A4,81,CB,43,28,14,CB,01,CB,09,1F,DC,A4
690 DATA 81,CB,4B,28,08,CB,01,CB,09,1F,DC,A4,81,10,E0,23
700 DATA 15,20,D7,C9,CD,2D,82,3E,80,C9,21,05,80,06,08,7E
710 DATA 2F,77,23,10,FA,C9,21,05,80,06,08,AF,77,23,10,FC
720 DATA C9,CB,59,20,06,CB,79,20,3E,18,04,CB,61,20,38,68
730 DATA 26,02,29,29,11,0D,80,06,04,7D,D3,E8,7C,D3,E9,D3
740 DATA EA,00,00,DB,E9,12,13,DB,E8,12,13,D3,EB,23,10,E9
750 DATA 11,05,80,0E,08,21,0D,80,06,08,AF,CB,06,1F,23,10
760 ' 8200-8245
770 DATA FA,12,13,0D,20,EF,C9,21,05,80,1E,02,AF,0E,04,CB
780 DATA 08,1F,1F,0D,20,F9,4F,87,81,0E,04,77,23,0D,20,FB
790 DATA 1D,20,E9,C9,7E,B7,C8,CD,2D,82,23,18,F7,F5,DB,40
800 DATA 0F,38,FB,F1,D3,10,F5,F3,3A,C1,E6,E6,FE,D3,40,F6
810 DATA 01,D3,40,FB,F1,C9
```

このプログラムを実行してから、

A＝&HF320　：　CALL　A(またはUSR関数)

と行なうと、画面コピーがとれます。なお、このプログラムはテキスト画面の文字のフォントが漢字ROMの1/4角と同じであることを利用しています。本書の画面コピーもこのプログラムを使ってとりました。

11-1-2　カラーグラフィック画面コピープログラム

次はグラフィック画面をコピーするプログラムを作ってみました。これは、カラーグラフィック画面を27×17cm(だいたいプリンタ用紙1枚分。PR201/101の場合は20×13cm)の大きさに引き伸ばしてコピーするもので、色に応じて濃淡がつけてあります。また縦横の比もほとんど正確で、プリンタの種類にもよりますが、1：1.03ぐらいになります。

使用法は、このプログラムを実行後、コピーしたいときに、

A＝&HF320　：　CALL　A

または　DEF　USR＝&HF320　：　DUMMY＝USR(0)

とします。プリンタの種類による変更は前と同じく、410行の最初のデータ(0E)を次のように変えます。(PC-8023(C)は使用できません。)

0E　…　PC-8821／8822，NM-9100／9200，NK-3618

16　…　PC-PR201/101

1E　…　NM-9300／9400(S)

また、410行の二番目のデータ(00)をFFに変更すると、白黒反転して(つまり画面と同じ)出力されます。

リスト　11-2

```
100 '
110 '  graphic copy
120 '
130 DEFINT A-Z
140 RESTORE *MAIN
150 FOR I=&H7000 TO &H714E
160   READ D$ : POKE I,VAL("&H"+D$)
170 NEXT
180 '
190 RESTORE *MOVM
200 FOR I=&HF320 TO &HF338
210   READ D$ : POKE I,VAL("&H"+D$)
220 NEXT
230 DEF USR=&HF320 : I=USR(0)
240 '
250 RESTORE *CALLER
260 FOR I=&HF320 TO &HF32F
270   READ D$ : POKE I,VAL("&H"+D$)
280 NEXT
290 '
300 END
310 '
320 *CALLER
330 DATA E5,DB,70,F5,3E,80,D3,70,CD,00,80,F1,D3,70,E1,C9
340 '
350 *MOVM
360 DATA 3A,C2,E6,F5,F6,02,F3,D3,31,21,00,70,11,00,80,01
370 DATA 00,03,ED,B0,F1,D3,31,FB,C9
```

```
380 '
390 *MAIN
400 ' 8000-807F
410 DATA 0E,00,C3,18,80,00,00,00,00,00,00,00,00,00,00,00
420 DATA 00,20,21,51,69,9E,DE,FF,DB,70,F5,3E,80,D3,70,21
430 DATA 23,81,11,00,83,01,0A,00,ED,B0,CD,70,80,DD,21,30
440 DATA FE,11,50,00,CD,C2,35,38,26,D5,CD,B1,80,06,C8,C5
450 DATA CD,C9,80,CD,F1,80,CD,07,81,11,B0,FF,DD,19,C1,10
460 DATA EE,CD,BF,80,11,81,3E,DD,19,D1,1B,7A,B3,20,D5,F1
470 DATA D3,70,21,69,80,CD,2D,81,C9,1B,41,1B,5D,0D,0A,00
480 DATA 3A,00,80,D6,0E,6F,0F,85,6F,26,00,11,8D,80,19,CD
490 ' 8080-80FF
500 DATA 2D,81,3E,1B,CD,36,81,3E,48,CD,36,81,C9,1B,54,31
510 DATA 36,1B,3E,00,00,00,00,00,00,1B,54,31,32,00,00,00
520 DATA 00,00,00,00,00,1B,3E,1A,41,1A,46,1B,54,31,36,00
530 DATA 00,21,B8,80,CD,2D,81,C9,1B,49,30,38,30,30,00,21
540 DATA C6,80,CD,2D,81,C9,0D,0A,00,0E,5E,16,03,CD,00,83
550 DATA 21,08,80,06,08,07,CB,16,23,10,FA,0D,15,20,EE,21
560 DATA 08,80,3A,01,80,4F,06,08,7E,A9,E6,07,77,23,10,F8
570 DATA C9,01,08,80,11,10,80,3E,08,08,0A,6F,26,00,19,7E
580 ' 8100-814E
590 DATA 02,03,08,3D,20,F3,C9,0E,04,21,08,80,1E,02,06,04
600 DATA CB,06,1F,CB,06,1F,23,10,F7,CD,36,81,1D,20,EF,0D
610 DATA 20,E7,C9,F3,ED,79,DD,7E,00,D3,5F,FB,C9,7E,B7,C8
620 DATA CD,36,81,23,18,F7,F5,DB,40,0F,38,FB,F1,D3,10,F5
630 DATA F3,3A,C1,E6,E6,FE,D3,40,F6,01,D3,40,FB,F1,C9
```

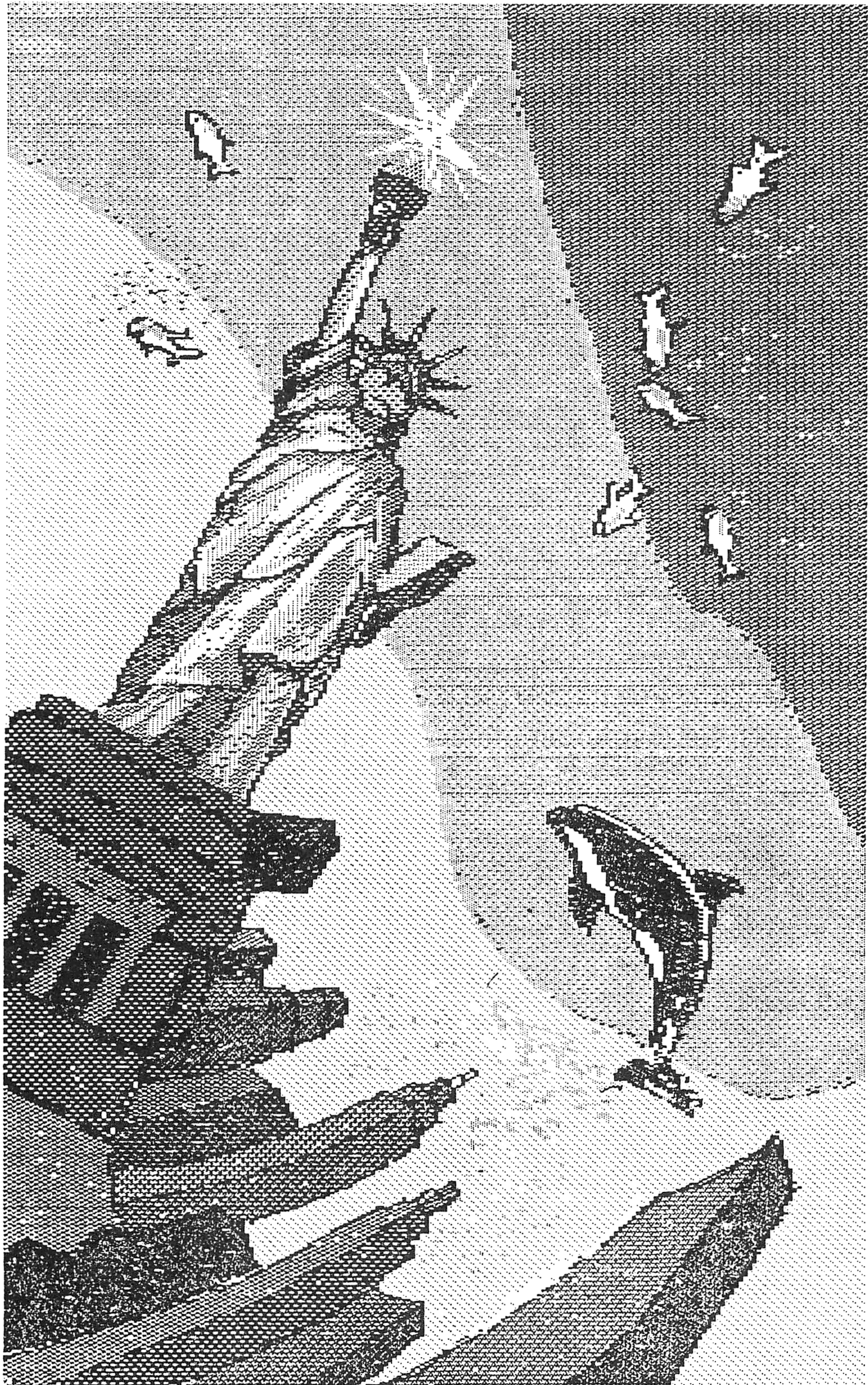
420行の初めの8個のデータは、画面上の各色1ドットに対して、プリンタに出力するドットパターンです。順にCOLOR 0～COLOR 7に対応しています。ひとつのデータ(パターン)は1バイトでできていて、このような構成になっています。

1	0
3	2
5	4
7	6

・数字はビット番号をあらわします。
・各マスはプリンタの1ドット
・全体で画面の1ドット

画面上の1ドットに対してプリンタは8ドット(2×4)打ち出しているわけで、これによって濃淡をつけています。打ちたいドットを1にして、ビットを順に並べ、16進数に変換し、420行の対応する色のデータを書き変えれば、好きなパターンで出力できます。

[コピー例]

ここで紹介したプログラムで出力した画面コピーの例をのせておきます。

11-2　プリント先を画面とプリンタで切り換える

PRINT文によって画面に出力するプログラムで、その出力先をプリンタに変更したいとき、プログラムリストをながめつつ、いちいちPRINT文をLPRINT文に直して行くというようなことは能率的ではありません。このようなときには第4章で述べたように”SCRN：”と”LPT1：”でファイルをオープンして出力するとよいのですが、ここでは、画面とプリンタで出力を切り換えることに的を絞って、もう少しいろいろと考えていきます。

11-2-1　CRTとプリンタへの出力をファイルとして扱う

まず、第4章で紹介した方法、つまり画面やプリンタをファイルとして扱う方法の実際の例をみてみます。

(1)ファイルディスクリプタを変える方法

画面とプリンタのどちらに出力するかによって、オープンするときのファイルディスクリプタを変えます。

CRTへ出力する場合

OPEN”SCRN：”FOR　OUTPUT　AS＃1

プリンタへ出力する場合

OPEN”LPT：”FOR　OUTPUT　AS＃1

処理が終わったら、忘れないようにCLOSE文を実行します。

この方法でメモリダンププログラムを作ってみましょう。

リスト　11-3

```
100 '
110 '    MEMORY DUMP  (I)
120 '
130 WIDTH 80,25
140 DEFINT A-Z
150 '
160 INPUT "START ADRS ? &H",SA$
170 INPUT "END   ADRS ? &H",EA$
180 '
190 SA=VAL("&H"+SA$) AND &HFFF0
200 EA=VAL("&H"+EA$)
210 '
220 INPUT "PRINTER (p) or CRT (c) ";OT$
230 IF OT$="p" THEN F$="LPT1:" ELSE IF OT$="c" THEN F$="SCRN:" ELSE 220
240 '
250 OPEN F$ FOR OUTPUT AS #1
260 '
270 FOR I=SA TO EA STEP 16
280  PRINT #1,RIGHT$("000"+HEX$(I),4)" : ";
290  FOR J=0 TO 15
300   PRINT #1,RIGHT$("0"+HEX$(PEEK(I+J)),2)" ";
310  NEXT J
320  PRINT #1,""
330 NEXT I
340 '
350 CLOSE
```

(2)ファイル番号を変える

別の方法として、CRTとプリンタに2つのファイルをオープンしておいて、ファイル番号を変数として使うことも考えられます。

OPEN "SCRN:" FOR OUTPUT AS#1

OPEN "LPT:" FOR OUTPUT AS#2

としておいて、CRTへ出力する場合

PRINT#1, ...

プリンタへ出力する場合

PRINT#2, ...

とします。

さきほどのプログラムを書きなおすとこうなります。

リスト 11-4

```
100 '
110 '    MEMORY DUMP  (II)
120 '
130 WIDTH 80,25
140 DEFINT A-Z
150 '
160 OPEN "LPT1:" FOR OUTPUT AS #1
170 OPEN "SCRN:" FOR OUTPUT AS #2
180 '
190 INPUT "START ADRS ? &H",SA$
200 INPUT "END   ADRS ? &H",EA$
210 '
220 SA=VAL("&H"+SA$) AND &HFFF0
230 EA=VAL("&H"+EA$)
240 '
250 INPUT "PRINTER (p) or CRT (c) ";OT$
260 IF OT$="p" THEN F=1 ELSE IF OT$="c" THEN F=2 ELSE 250
270 '
280 FOR I=SA TO EA STEP 16
290  PRINT #F,RIGHT$("000"+HEX$(I),4)" : ";
300  FOR J=0 TO 15
310   PRINT #F,RIGHT$("0"+HEX$(PEEK(I+J)),2)" ";
320  NEXT J
330  PRINT #F,""
340 NEXT I
350 '
360 CLOSE
```

11-2-2 PRINT to LPRINTコマンドを作る

新しく作るプログラムに対しては、上のテクニックを使うとよいことがわかりました。それでは、今までに作っていたPRINT文を使ったプログラムはどうしますか。何とかしてみたいところです。

そこで、次の実験をしてみましょう。

リスト 11-5

```
100 POKE &HE64C,1 : PRINT ABC"
```

上のプログラムを実行すると、文字が画面ではなくプリンタに出力されます。

POKE文を実行しただけですが、このE64CH番地というのはなんでしょう。実はこれは、

1文字出力をどこに対して行なうかを表わすワークエリアの番地なのです。ここの内容が0であれば画面、そうでなければプリンタに出力されます。

ここで、「なぁーんだ、それじゃプリンタに出したいときはPOKE & HE64C, 1をやるだけでいいんだな。」と思ってしまってはいけないのです。次のプログラムを実行してみて下さい。

リスト 11-6

```
100 POKE &HE64C,1 : PRINT ABC"
110 PRINT "DEF"
```

110行では画面に出力されてますね。これは、E64CHというワークエリアには、1度PRINT文が実行された後は、0が書き込まれてしまうためです。

そこで、次のプログラムを実行して見て下さい。

リスト 11-7

```
100 '
110 '    PRINT to LPRINT
120 '
130 FOR I=&HF260 TO &HF286
140  READ D$ : POKE I,VAL("&H"+D$)
150 NEXT I
160 '
170 CH=&HEEA7:POKE CH,&HC3:POKE CH+1,&H60:POKE CH+2,&HF2
180 PH=&HEDCF:POKE PH,&HC3:POKE PH+1,&H7A:POKE PH+2,&HF2
190 '
200 DATA FE,95,28,0C,FE,EE,C2,93,03,D7,C2,93,03,AF,18,06
210 DATA D7,C2,93,03,3E,01,32,86,F2,C9,F5,3A,86,F2,B7,28
220 DATA 03,32,4C,E6,F1,C9,00
```

何も変化はありませんが、あなたのN88-BASICにはPOLLという便利なコマンドが追加されたわけです。

試しに次のようにやってみて下さい。

リスト 11-8

```
poll on
Ok
print "ABCDEF"
Ok
```

PRINT文でプリンタへ文字が出力されましたね。うまく動かなかったら上のプログラムをもう一度確かめて実行して下さい。

次にもとに戻します。

リスト 11-9

```
poll off
Ok
print "ABCDEF"
ABCDEF
Ok
```

今度は、画面に出力されますね。

さあ準備は整いました。前のプログラムを、このコマンドを使って書き直してみましょう。

リスト　11-10

```
100 '
110 '    MEMORY DUMP  (III)
120 '
130 WIDTH 80,25
140 DEFINT A-Z
150 '
160 INPUT "START ADRS ? &H",SA$
170 INPUT "END   ADRS ? &H",EA$
180 '
190 SA=VAL("&H"+SA$) AND &HFFF0
200 EA=VAL("&H"+EA$)
210 '
220 INPUT "PRINTER (p) or CRT (c) ";OT$
230 IF OT$="p" THEN POLL ON ELSE IF OT$="c" THEN POLL OFF ELSE 220
240 '
250 FOR I=SA TO EA STEP 16
260  PRINT RIGHT$("000"+HEX$(I),4)" : ";
270  FOR J=0 TO 15
280   PRINT RIGHT$("0"+HEX$(PEEK(I+J)),2)" ";
290  NEXT J
300  PRINT
310 NEXT I
```

11-3　漢字プリンタ

11-3-1　使って便利な漢字←→キャラクタ対応表

漢字の出力には、漢字JISコード表を用います。これは、１つの漢字に対して２バイトの16進数を与えるものです。

このJISコード表を使って漢字列を印字するプログラムは、次のようになります。

リスト　11-11

```
100 KI$=CHR$(27)+"K" : KO$=CHR$(27)+"H"
110 '
120 LPRINT KI$;
130 FOR I=1 TO 8
140   READ HD,LD
150   LPRINT CHR$(HD);CHR$(LD);
160 NEXT
170 LPRINT KO$
180 '
190 DATA &H46,&H7C,&H4B,&H5C,&H45,&H45,&H35,&H24
200 DATA &H33,&H74,&H3C,&H30,&H32,&H71,&H3C,&H52
```

日本電気株式会社

ところがこれは、次のようにしてもよいわけです。

```
100 KI$=CHR$(27)+"K" : KO$=CHR$(27)+"H"
110 '
120 LPRINT KI$; "F¦K¥EE5$3t<02q<R"; KO$
130 '
```

日本電気株式会社

これは、漢字JISコードをキャラクタ2文字に置き換えて、直接LPRINT文中に入れたものですが、こうするとプログラムも短くなるし、データ文として持ったときも1／5になります。ただ難点としては、漢字JISコードからキャラクタへ変換する手間が必要です。

そこで、この手間を省くために、漢字からキャラクタを得る一覧表を紹介します。

この表は次のようなものです。

リスト　11-13

```
〔 ア 〕 ---------------------------------------------------------
        亜=0!  唖=0「 娃=0#  阿=0$  哀=0%  愛=0&  挨=0'  姶=0(  逢=0)  葵=0*
        茜=0+  穐=0,  悪=0-  握=0.  渥=0/  旭=00  葦=01  芦=02  鯵=03  梓=04
        圧=05  斡=06  扱=07  宛=08  姐=09  虻=0:  飴=0;  絢=0<  綾=0=  鮎=0>
        或=0?  粟=0@  袷=0A  安=0B  庵=0C  按=0D  暗=0E  案=0F  闇=0G  鞍=0H
        杏=0I
〔 イ 〕 ---------------------------------------------------------
        以=0J  伊=0K  位=0L  依=0M  偉=0N  囲=0O  夷=0P  委=0Q  威=0R  尉=0S
        惟=0T  意=0U  慰=0V  易=0W  椅=0X  為=0Y  畏=0Z  異=0〔 移=0¥  維=0〕
        緯=0^  胃=0_  萎=0═ 衣=0a  謂=0b  違=0c  遺=0d  医=0e  井=0f  亥=0g
        域=0h  育=0i  郁=0j  磯=0k  一=0l  壱=0m  溢=0n  逸=0o  稲=0p  茨=0q
        芋=0r  鰯=0s  允=0t  印=0u  咽=0v  員=0w  因=0x  姻=0y  引=0z  飲=0{
        淫=0¦  胤=0}  蔭=0~  院=1!  陰=1「 隠=1#  韻=1$  吋=1%
〔 ウ 〕 ---------------------------------------------------------
        右=1&  宇=1'  烏=1(  羽=1)  迂=1*  雨=1+  卯=1,  鵜=1-  窺=1.  丑=1/
        碓=10  臼=11  渦=12  嘘=13  唄=14  欝=15  蔚=16  鰻=17  姥=18  厩=19
        浦=1:  瓜=1;  閏=1<  噂=1=  云=1>  運=1?  雲=1@
〔 エ 〕 ---------------------------------------------------------
        荏=1A  餌=1B  叡=1C  営=1D  嬰=1E  影=1F  映=1G  曳=1H  栄=1I  永=1J
        泳=1K  洩=1L  瑛=1M  盈=1N  穎=1O  頴=1P  英=1Q  衛=1R  詠=1S  鋭=1T
        液=1U  疫=1V  益=1W  駅=1X  悦=1Y  謁=1Z  越=1〔 閲=1¥  榎=1〕 厭=1^
        円=1_  園=1═ 堰=1a  奄=1b  宴=1c  延=1d  怨=1e  掩=1f  援=1g  沿=1h
        演=1i  炎=1j  焔=1k  煙=1l  燕=1m  猿=1n  縁=1o  艶=1p  苑=1q  薗=1r
        遠=1s  鉛=1t  鴛=1u  塩=1v
```

例えば、'愛'という漢字に対しては、'0＆'という文字列が得られます。複数の漢字については、上の例のように、文字列をつないでLPRINTしてやればよいわけです。

この表にも少し欠点があります。JISコード(外字を除く全角文字)では1バイトに現われる値は21H～7EHです。ところが22H(")は文字列の中に書くことはできませんし、60Hは見た目にはスペースと区別がつきません。この問題はPC-8822の場合はちょっとしたテクニックで解決できます。この2つの文字はその値に80Hを足した値(ビット7を1にする)に対応する文字を出すとうまくいきます。つまり22HのときはA2Hに対応する文字(｢)をプリントします。60Hのときは＝(グラフィックの＝) です。入力するときにこれらの文字を使うと、文字列の中に入れることができます。たとえば「劇」という文字を印字するときには、(プリンタが漢字モードになっているとして)

LPRINT　"7＝"

とします。

しかし、PC-PR201/101の場合はこの方法はうまくいきません。CHR$を使うしかなさそうです。(60HはファンクションキーにCHR$(&h60)を定義するという方法もあります。)

この表の完全なものは付録にありますが、自分なりにアレンジした表を作りたいという人のために、この表を出力するプログラムをあげておきます。

リスト 11-14 非漢字←→キャラクタ

```
100 '
110 '   ｶﾝｼﾞ ﾌﾟﾘﾝﾀ   ｷｺﾞｳ ｷｬﾗｸﾀ ﾋｮｳ
120 '
130 DEFINT A-Z
140 '
150 LPRINT CHR$(27)"H";
160 LC=0
170 LPRINT "〔 ｷｺﾞｳ 〕 "STRING$(52,"-"):LPRINT SPC(6);
180 FOR I=&H2121 TO &H2770
190  CH=I¥256 : CL=I MOD 256
200  IF CL=&H7F THEN I=(CH+1)*256+&H21 : GOTO 190
210  LPRINT CHR$(27)"K"CHR$(CH)CHR$(CL);
220  IF CH=&H22 THEN CH=&HA2
230  LPRINT CHR$(27)"H="CHR$(CH);
240  IF CL=&H60 THEN CL=&HE0
250  IF CL=&H22 THEN CL=&HA2
260  LPRINT CHR$(CL)"  ";
270  LC=LC+1
280  IF LC=10 THEN LC=0:LPRINT : LPRINT SPC(6);
290 NEXT
```

リスト 11-15 漢字←→キャラクタ

```
1000 '
1010 '   ｶﾝｼﾞ ﾌﾟﾘﾝﾀ   ｶﾝｼﾞ ｷｬﾗｸﾀ ﾋｮｳ
1020 '
1030 DEFINT A-Z
1040 '
1050 LPRINT CHR$(27)"H";
1060 CCE=&H3021
1070 FOR HR=ASC("ｱ") TO ASC("ﾜ")
1080  CCS=CCE
1090  READ CCE
1100  LPRINT "〔 "CHR$(HR)" 〕 "STRING$(54,"-"):LPRINT SPC(6);
1110  LC=0
1120  FOR I=CCS TO CCE-1
1130   CH=I¥256 : CL=I MOD 256
1140   IF CL=&H7F THEN I=(CH+1)*256+&H21 : GOTO 1130
1150   LPRINT CHR$(27)"K"CHR$(CH)CHR$(CL);
1160   LPRINT CHR$(27)"H="CHR$(CH);
1170   IF CL=&H60 THEN CL=&HE0
1180   IF CL=&H22 THEN CL=&HA2
1190   LPRINT CHR$(CL)"  ";
1200   LC=LC+1
1210   IF LC=10 THEN LC=0:LPRINT : LPRINT SPC(6);
1220  NEXT I
1230  LPRINT
1240 NEXT
1250 '
1260 DATA &H304a,&H3126,&H3141,&H3177
1270 DATA &H323c,&H346b,&H3665,&H3735,&H3843
1280 DATA &H3a33,&H3b45,&H3f5a,&H4024,&H4139
1290 DATA &H423e,&H434d,&H4445,&H4462,&H4546
1300 DATA &H4660,&H4673,&H4728,&H4729,&H4735
1310 DATA &H4743,&H485b,&H4954,&H4a3a,&H4a5d
1320 DATA &H4b60,&H4c23,&H4c33,&H4c3d,&H4c4e
1330 DATA &H4c69,&H4c7b,&H4d3d
1340 DATA &H4d65,&H4d78,&H4e5c,&H4e61,&H4f24
1350 DATA &H4f41,&H4f54
```

11-3-2　外字データ作成プログラム

漢字プリンタには外字機能があります。ここでは、その外字データを作成する簡単なエディタと、ディスクファイル上の外字データを漢字プリンタにロードするローダーを紹介します。

まず次のプログラムで、外字データファイルを初期化します。プリンタが24ピンの場合は130行のSIZE＝16をSIZE＝24に変更してから実行してください。

ドライブ１のディスクにGAIJI.16またはGAIJI.24というファイル名で外字データファイルが作られます。

○外字デ-タファイル初期化プログラム

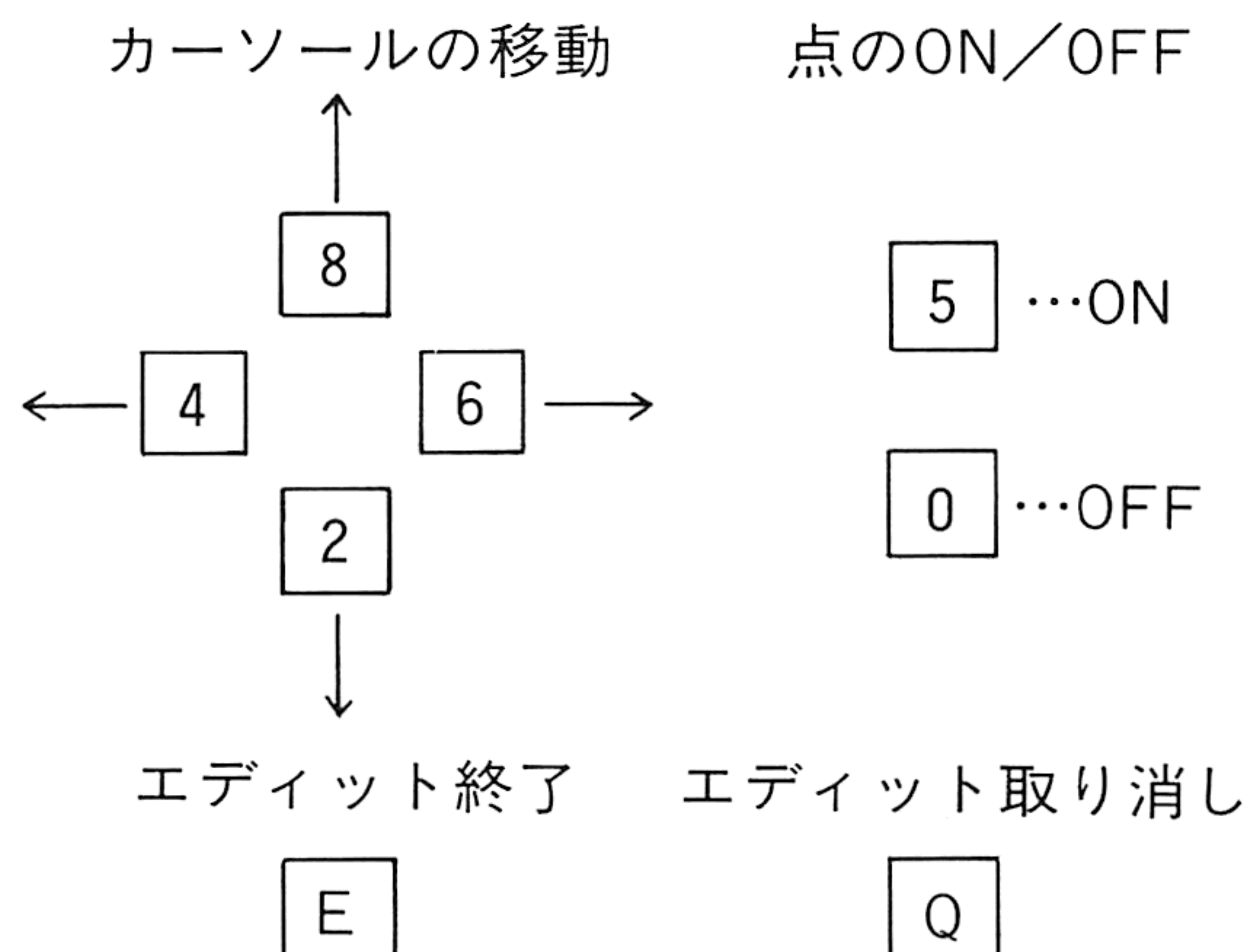
リスト　11-16

```
100 '
110 '      ｶﾝｼﾞ ﾌﾟﾘﾝﾀ ｶﾞｲｼﾞ ﾃﾞｰﾀ ﾌｧｲﾙ ｲﾆｼｬﾗｲｽﾞ
120 '
130 SIZE = 16    :' ｺｺｦ 16 ﾏﾀﾊ 24 ﾆ ｶｴﾃｸﾀﾞｻｲ
140 GOSUB *OPEN.FILE
150 '
160 CONSOLE 0,25,1,0 : WIDTH 40,25
170 LOCATE 3,10
180 PRINT "ｶﾝｼﾞ ﾌﾟﾘﾝﾀ ｶﾞｲｼﾞ ﾃﾞｰﾀ ﾌｧｲﾙ ｲﾆｼｬﾗｲｽﾞ"
190 '
200 LSET DT$=STRING$(72,0)
210 FOR I=64 TO 1 STEP -1
220   PUT #1,I
230 NEXT I
240 '
250 END
260 '
270 *OPEN.FILE
280  OPEN "GAIJI."+STR$(SIZE) AS #1
290  FIELD #1,72 AS DT$
300 RETURN
```

この後、次のプログラムにより、外字コード7620H～765FHの64文字(PC-PR201/101の場合は7620H-7657Hの56文字だけを使用してください)の外字のエディットができます。このプログラムも、24ピンプリンタの場合はSIZE＝24に変更してください。

最初に外字コードを入力すると、その外字のパターンが画面上に現われます。「フォントを作成してください」というメッセージがでたら、テンキーとスペースキーで文字フォントを作成して下さい。キー操作は、次のとおりです。

図11-3-A　キー操作

カーソールの移動		点のON／OFF	
8	↑	5	…ON
4	←	0	…OFF
6	→		
2	↓		
エディット終了	E	エディット取り消し	Q

○エディタ・プログラム

リスト 11-17

```
1000 '
1010 '  ｶﾝｼﾞ ﾌﾟﾘﾝﾀ ｶﾞｲｼﾞ ﾃﾞｰﾀ ﾌｧｲﾙ ｻｸｾｲ
1020 '
1030 SCREEN 0,3 : ROLL 197
1040 DEFINT A-Z
1050 '
1060 SIZE = 16  : ' ｺｺｦ 16 ﾏﾀﾊ 24 ﾆ ｼﾃｸﾀﾞｻｲ
1070 GOSUB *OPEN.FILE
1080 '
1090 YBYTE = SIZE¥8
1100 MAXBYTE = SIZE*YBYTE - 1
1110 '
1120 CONSOLE ,,0,1 : WIDTH 40,25
1130 DIM DT(MAXBYTE)
1140 '
1150 CLS
1160 LOCATE 0,10 : PRINT SPACE$(40)
1170 LOCATE 0,10 : INPUT "ｶﾞｲｼﾞ ｺｰﾄﾞ ( &H7620-&H765F )";K$
1180 IF LEFT$(K$,1)="&" THEN G.CODE = VAL(K$) ELSE G.CODE = VAL("&H"+K$)
1190 IF G.CODE<&H7620 OR G.CODE>&H765F THEN 1160
1200 '
1210 DT=G.CODE-&H761F
1220 '
1230 CLS : COLOR 4
1240 PRINT "  ┌────┐"
1250 PRINT "  │    │"
1260 PRINT "  │    │"
1270 PRINT "  └────┘"
1280 COLOR 7
1290 KJ$="03z;!!n:.@" : PX=48 : PY=8 : GOSUB *PUT.KANJI : SCREEN ,0
1300 COLOR 4 : LOCATE 0,7 : PRINT "CODE : &H";HEX$(G.CODE)
1310 'PRINT
1320 LOCATE 15,0 : COLOR 5
1330 IF SIZE = 16 THEN  PRINT "0123456789ABCDEF"
1340 IF SIZE = 24 THEN  PRINT "012345678901234567890123"
1350 FOR I=0 TO SIZE-1
1360  LOCATE 13,I+1 : COLOR 5 : PRINT STR$(I MOD 10);
1370  COLOR 7 : PRINT STRING$(SIZE,"·");
1380 NEXT
1390 GOSUB *P.DATA
1400 '
1410 KJ$="U%)%s%H%r$n:.@7$F$/$@$5$$$": PX=0 : PY=168 : GOSUB *PUT.KANJI
1420 CX=15 : CY=1
1430 '
1440 LOCATE CX,CY
1450 KY$=INPUT$(1)
1460 ON INSTR(" 246850EeQq",KY$)
     GOSUB *SP,*DW,*LF,*RT,*UP,*ST,*RS,*EN,*EN,*QU,*QU
1470 GOTO 1440
1480 '
1490 *DW : IF CY<SIZE THEN CY=CY+1
1500  RETURN
1510 *LF : IF CX>15  THEN CX=CX-1
1520  RETURN
1530 *RT : IF CX<SIZE+14 THEN CX=CX+1
1540  RETURN
1550 *UP : IF CY>1  THEN CY=CY-1
1560  RETURN
1570 *ST : PRINT "●";
1580  RETURN
1590 *RS : PRINT "·";
1600  RETURN
1610 *SP
1620  IF PEEK(&HF3C8+120*CY+2*CX) = ASC("●") THEN PRINT "·"; ELSE PRINT "●";
1630 RETURN
1640 *EN
```

```
1650  LINE (PX,PY)-STEP(LEN(KJ$)*16,16),0,BF
1660  KJ$="3$l$G$$$$$G$9$+$" : PX=0 : PY=168 : GOSUB *PUT.KANJI
1670  LOCATE 0,23 : PRINT "(y/n):"; :S$=INPUT$(1)
1680  IF S$<>"y" AND S$<>"n" THEN 1670
1690  LINE (PX,PY)-STEP(LEN(KJ$)*16,16),0,BF
1700  LOCATE 0,23 : PRINT SPACE$(14);
1710  IF S$="n" THEN RETURN
1720  '
1730  GOSUB *G.DATA
1740  KJ$="LJN$z;r$n:.@7$^$9$+$!!" : PX=0 : PY=168 : GOSUB *PUT.KANJI
1750  LOCATE 0,23 : PRINT "(y/n):"; : S$=INPUT$(1)
1760  IF S$<>"y" AND S$<>"n" THEN 1750
1770  SCREEN ,3 : ROLL 197 : SCREEN ,0
1780  IF S$="y" THEN 1150
1790 *QU
1800  CLS : SCREEN ,3 : ROLL 197 : SCREEN ,0
1810 END
1820 '
1830 *OPEN.FILE
1840  OPEN "GAIJI."+STR$(SIZE) AS #1
1850  FIELD #1,24*3 AS DT$
1860 RETURN
1870 '
1880 *P.DATA
1890  KJ$="7$P$i$/$*$TBA$/$@$5$$$": PX=0 : PY=168 : GOSUB *PUT.KANJI
1900  GET #1,DT
1910  FOR I=0 TO MAXBYTE
1920   DT(I)=ASC(MID$(DT$,I+1,1))
1930  NEXT I
1940  FOR I=0 TO SIZE-1
1950   FOR J=0 TO YBYTE-1
1960    CN=I*YBYTE+J
1970    FOR K=0 TO 7
1980     IF DT(CN) MOD 2=1 THEN LOCATE 15+I,1+J*8+K : PRINT "●";
1990     DT(CN)=DT(CN)¥2
2000    NEXT
2010   NEXT
2020  NEXT
2030  LINE (PX,PY)-STEP(LEN(KJ$)*16,16),0,BF
2040 RETURN
2050 '
2060 *G.DATA
2070  KJ$="7$P$i$/$*$TBA$/$@$5$$$": PX=0 : PY=168 : GOSUB *PUT.KANJI
2080  FOR I=0 TO SIZE-1
2090   IP = I*2 + &HF3C8 + 15*2 + 120
2100   FOR J=0 TO YBYTE-1
2110    JP = (J*8)*120 + IP
2120    CN=I*YBYTE+J
2130    AP=1
2140    FOR K=0 TO 7
2150     IF PEEK(JP+K*120)=ASC("●") THEN DT(CN)=DT(CN)+AP
2160     AP=AP+AP
2170    NEXT
2180   NEXT
2190  NEXT
2200  AP$=""
2210  FOR I=0 TO MAXBYTE
2220   AP$=AP$+CHR$(DT(I))
2230  NEXT I
2240  LSET DT$=AP$
2250  PUT #1,DT
2260  LINE (PX,PY)-STEP(LEN(KJ$)*16,16),0,BF
2270 RETURN
2280 '
2290 *PUT.KANJI
2300  FOR KJ=0 TO LEN(KJ$)-1 STEP 2
2310    PUT (PX+KJ*8,PY),KANJI(CVI(MID$(KJ$,KJ+1,2))),PSET
2320  NEXT
2330 RETURN
```

作成された外字データファイルは、次のプログラムを使って漢字プリンタにロードできます。同じく24ピンプリンタではSIZE＝24に変更してください。またPC-PR201/101では250行のFOR I=1 TO 64をFOR I= 1 TO 56に変更してください。

リスト 11-18外字データローダー・プログラム

```
100 '
110 '   ｶﾝｼﾞ ﾌﾟﾘﾝﾀ ｶﾞｲｼﾞ ﾃﾞｰﾀ ﾛｰﾄﾞ
120 '
130 DEFINT A-Z
140 '
150 SIZE = 16  :' ｺｺｦ 16 ﾏﾀﾊ 24 ﾆ ｶｴﾃｸﾀﾞｻｲ
160 '
170 GOSUB *OPEN.FILE
180 WIDTH LPRINT 255
190 '
200 CONSOLE ,,1,0 : WIDTH 40,25
210 '
220 LOCATE 3,10
230 PRINT "ｶﾝｼﾞ ﾌﾟﾘﾝﾀ ｶﾞｲｼﾞ ﾃﾞｰﾀ ﾛｰﾄﾞ  (";SIZE;"ﾋﾟﾝ)"
240 '
250 FOR I=1 TO 64
260  GOSUB *P.DATA
270 NEXT I
280 '
290 END
300 '
310 *OPEN.FILE
320  OPEN "GAIJI."+STR$(SIZE) AS #1
330  FIELD #1,72 AS DT$
340 RETURN
350 '
360 *P.DATA
370  GET #1,I
380  LOCATE 9,13
390  PRINT "Loading .. &H";HEX$(&H761F+I)
400  LPRINT CHR$(27);
410  IF SIZE=16 THEN LPRINT "*";  ELSE LPRINT "+";
420  LPRINT CHR$(&H76);CHR$(&H1F+I);
430  LPRINT LEFT$(DT$,SIZE*SIZE¥8);
440  LPRINT CHR$(4);
450 RETURN
```

外字データファイルは、”GAIJI.16”または”GAIJI.24”で統一されていますが、必要があれば、*OPEN.FILEサブルーチンを変えて下さい。

11-4　WIDTH LPRINTとTABコード

N88-BASICでは、プリンタの印字桁数を制御するために、WIDTH　LPRINT文があります。ここでは、WIDTH　LPRINT文の、マニュアルにない使い方を見てみましょう。

11-4-1　WIDTH LPRINTの値と出力

WIDTH　LPRINTの値と、その出力について実際に見てみましょう。(マニュアルとは違った動作をするところがありますので注意が必要です。)

動作のチェックのために次のプログラムを用います。(プリンタは、80桁のものを使用)

リスト　11-19

```
100 INPUT LW
110 WIDTH LPRINT LW
120 FOR I=1 TO 300
130   LPRINT "*";
140 NEXT I
```

①LW　(WIDTH LPRINTの値)がプリンタの幅と同じかまたは小さい場合(LW＝60)

リスト　11-20

```
************************************************************
************************************************************
************************************************************
************************************************************
************************************************************
```

この場合は、60文字印字したところで改行しています。

②LWがプリンタの幅より大きい場合(LW＝100)

リスト　11-21

```
********************************************************************************
********************
********************************************************************************
********************
********************************************************************************
********************
```

この場合は、80文字で改行(プリンタ側で行なわれる)した後、20文字めで再び改行しています。PC本体から見れば100文字めで改行したことになりますね。

普通こういった使い方はしないでしょう。

③LWが０の場合

リスト　11-22

```
*

*

*
```

マニュアルでは、LWが0の時は、256と解釈されるとなっていますが、実際には、１文字印字して2回改行することの繰り返しになります。

これも使える値ではないようですね。

④LWが255の場合

リスト　11-23

```
********************************************************************************
********************************************************************************
********************************************************************************
************************************************************
```

マニュアルにはないのですが、LWが255の時は、特別の働きをします。一見②と同じようになると思えますが、実際は、WIDTH LPRINTの値は無視されることになります。つまり、改行はプリンタまかせということです。

このことは、不特定の桁数のプリンタ用のプログラムを作成する時に意味があります。

なお、リセット時には、この値は255となっています。

おなじプログラムでも、プリンタ出力が異なる現象が起こるのは、この値が違っている場合が多いようです。

プリンタ用のソフトには、念のために、WIDTH LPRINT 255を入れておくことをおすすめします。

11-4-2　水平タブコードの出力とドット対応グラフィック

WIDTH LPRINT 255にはもう１つの機能があります。

次のプログラムを実行すると、画面上では、TABコードが正常に出力されますが、プリンタでは、次のように異なったものが出力されます。

リスト　11-24

```
100 WIDTH LPRINT 255
110 TAB.CODE$=CHR$(9)
120 PRINT "ABC";TAB.CODE$;"DEF"
130 LPRINT "ABC";TAB.CODE$;"DEF"
```

プリンタ：ABCDEF

画　　面：ABC　　　DEF

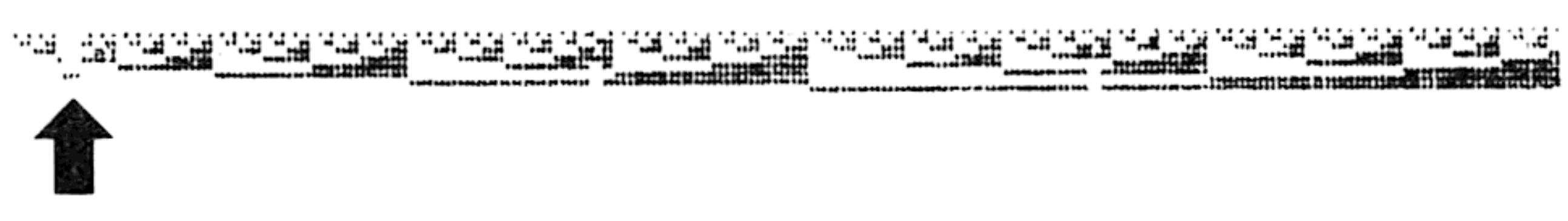
ここでWIDTH LPRINTの値を80にすると、プリンタにも画面と同じ出力が得られますね。これはどういうことかというと、N88-BASICでは、TABコードの出力については、POS、LPOSの値を参照して、必要な数だけのCHR$(32)を出力するという方法をとっています。

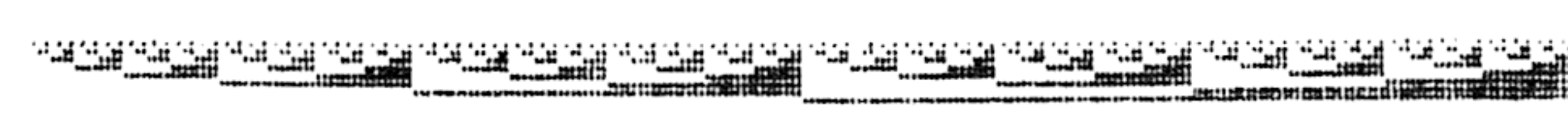
ところがWIDTH LPRINTの値が255のときは、このようなことはせずに、プリンタに対してCHR$(9)をそのまま送ります。そこでプリンタ側での水平タブ位置が設定されていない場合、上のように一見無視された形になるわけです。

逆に言えば、N88-BASICでは、CHR$(9)を送るのにプリンタポートを直接コントロールするという面倒な手順を使わなくても、水平タブコードを送ることが可能になったということです。

N-BASICモードや、WIDTH LPRINTの値が255以外のとき、次のプログラムを実行すると、CHR$(9)の出力に対して、３個のCHR$(32)がプリンタに送られ、(A)のような出力になります。

リスト　11-25

```
100 LPRINT CHR$(27);"S0256";
110 FOR I=0 TO 255
120  LPRINT CHR$(I);
130 NEXT
```

これを避けるために、従来はプリンタポ-トを直接コントロールするプログラムを作っていたわけですが、WIDTH LPRINT 255によって、このような心配も不要になります。

WIDTH LPRINT 255を実行した後、上のプログラムを実行して見て下さい。目的とした出力(B)が得られます。

11-5　I／O　ポートアドレス

プリンタを機械語などで直接コントロールしたいときのために、プリンタポートの内容と使用法を載せておきます。

プリンタコントロールポート

(1)　プリンタ出力データ

OUT　10H

7	6	5	4	3	2	1	0
PDB 7	PDB 6	PDB 5	PDB 4	PDB 3	PDB 2	PDB 1	PDB 0

(2)プリンタストローブ

OUT　40H

7	6	5	4	3	2	1	0
							PSTB

PSTB　　プリンタへのストローブ

0　：　ストローブ　オン

1　：　ストローブ　オフ

OUT　40Hポートは、プリンタストローブの他に全てのビットを使用していますので、注意して下さい。

○使用法

プリンタに1文字(1バイト)を出力する手順は次のようになります。

①プリンタがREADYになるまでループします。

②データをプリンタデータポートに出力します。

③$\overline{\text{PSTB}}$を０にします。

④$\overline{\text{PSTB}}$を１にします。

③，④で$\overline{\text{PSTB}}$をコントロールするときに、はポート40H(出力)の他のビットを変更しないように、E6C1H番地(N$_{88}$-BASICのとき)にある、ポート40Hに出力した内容を使います。

この手順をN$_{88}$-BASICで書いたサブルーチンをあげます。変数Aに入っている文字コードを出力します。

```
100  *PRINTER.OUT
110  WHILE  INP(&H40) AND 1 : WEND
120  OUT &H10,  A
130  OUT &H40,  PEEK(&HE6C1) AND &HFE
140  OUT &H40,  PEEK(&HE6C1) OR 1
150  RETURN
```

○使用例

機械語でAからZまでを出力させるプログラム例を示します。

リスト　11-28

```
                             ORG   0F320H

F320 3E41                    LD    A, 'A'           ; START WITH 'A'
F322 061A                    LD    B, 26            ; 26 CHARS
F324               NEXT:
F324 CD35F3                  CALL  OUTCHR           ; OUTPUT A CHAR
F327 3C                      INC   A
F328 10FA                    DJNZ  NEXT             ; LOOP 26 TIMES
F32A 3E0D                    LD    A, 13
F32C CD35F3                  CALL  OUTCHR           ; OUTPUT CR
F32F 3E0A                    LD    A, 10
F331 CD35F3                  CALL  OUTCHR           ; OUTPUT LF
F334 C9                      RET

F335               OUTCHR:
F335 F5                      PUSH  AF               ; SAVE THE CHAR
F336               OUTC1:
F336 DB40                    IN    A, (40H)
F338 0F                      RRCA
F339 38FB                    JR    C, OUTC1         ; LOOP WHILE BUSY

F33B F1                      POP   AF               ; RESTORE THE CHAR
F33C D310                    OUT   (10H), A         ; OUT TO PRINTER
F33E F5                      PUSH  AF

F33F F3                      DI
F340 3AC1E6                  LD    A, (0E6C1H)      ; PORT 40H BACKUP
F343 E6FE                    AND   0FEH
F345 D340                    OUT   (40H), A         ; PSTB ON
F347 F601                    OR    1
F349 D340                    OUT   (40H), A         ; PSTB OFF
F34B 32C1E6                  LD    (0E6C1H), A
F34E FB                      EI
```

```
F34F F1                 POP   AF
F350 C9                 RET
```

このプログラムを実行しますと、

ABCDEFGHIJKLMNOPQRSTUVWXYZ

とプリンタに出力されます。

第12章　カセットインタフェース

カセットインタフェースは、シリアルインタフェース用LSI μPD8251を用い、RS-232Cインタフェースと切り換えて使います。

12-1　データフォーマット

12-1-1　N88-BASICプログラムファイル

N88-BASICプログラムをカセットにSAVEした時のフォーマットは、次のようになります。

WAIT 2秒	D3×10回				ファイルネーム6バイト						WAIT 0.1秒
Mark	D3	D3	········	D3	"T"	"E"	"S"	"T"	00	00	Mark

プログラム本体	00×9回				WAIT 1秒
(E658,59) ~ (EB18,19) − 1	00	00	········	00	Mark

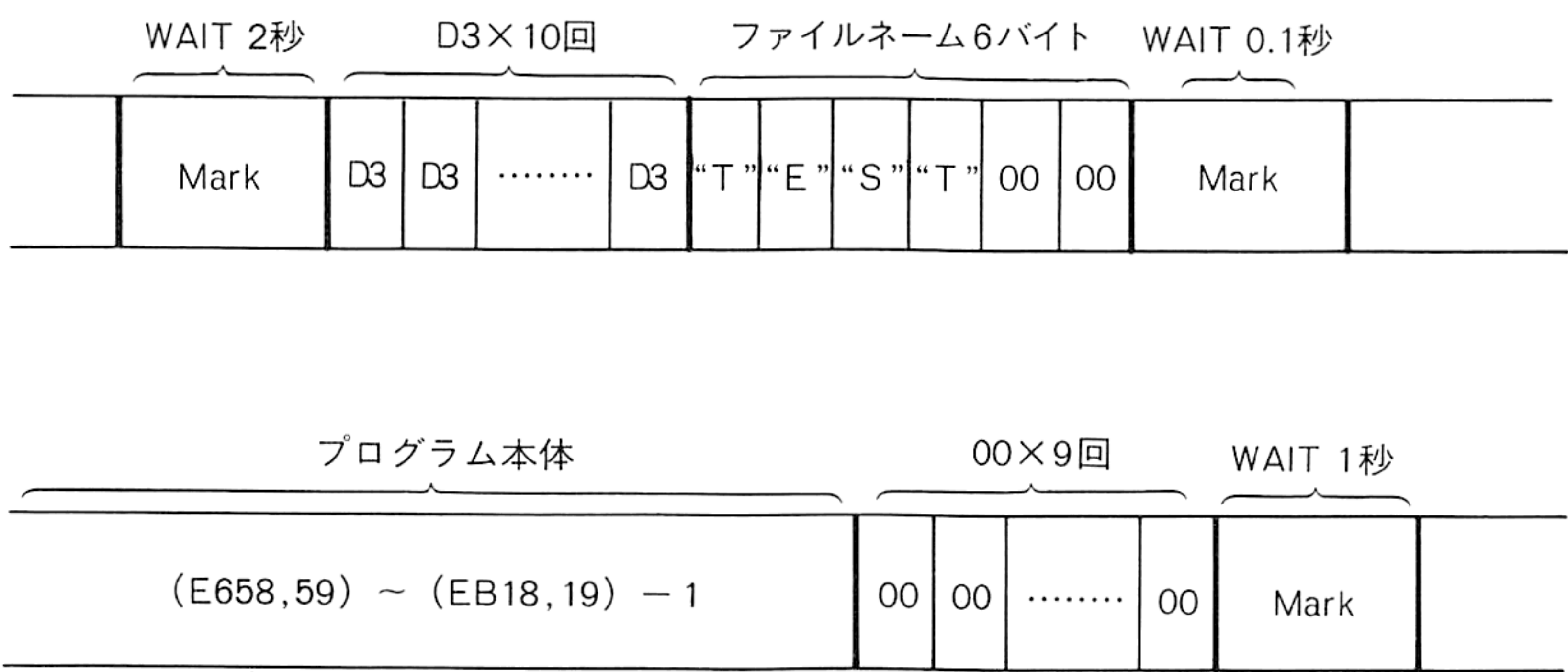



図12-1-A　プログラムファイルフォーマット

まずモータをONにし、テープ走行が安定するまで待ったあと、D3Hを10回書き込み、6バイトのファイルネームを書き込みます。ファイルネームが6文字未満のときは、その後に00Hをつけ加え、6バイトにします。その後再びWAITがありますが、これは、LOADした時、この時間を利用して'Found'や'Skip'などの表示をしているためで、これがないとオーバーランしてしまう恐れがあります。

そして、プログラム本体です。プログラム本体は、E658, 59H番地に入っているプログラム開始番地から、EB18, 19H番地に入っている番地の1つ前までを書き込みます。この本体の最後の3バイトは00Hになっていて、次に続く9回の00Hと合わせ、12回の00Hでエンドマークを作ります。

LOADの時、D3Hを10回読むとプログラムファイルと見なし、ファイルネームの比較を行ないます。一致していればプログラムを読み込み、00Hが10回連続して読み出されるとプログラムファイルの終了と見なし、LOADを終了します。

ところで、このフォーマットは、N-BASICのフォーマットと全く同一です。事実、メモリ容量を超えない範囲で、ボーレイトを600ボー(cas2)にした時、N88モードでカセットにSAVEしたプログラムをN-BASICモードで読むことは可能ですし、その逆も可能です。しかし、いずれの場合も、実行することはできません。なぜなら、中間言語のコードが違って

いますし、ステートメントの機能も一部異なっているからです。

12-1-2　N88-BASICデータファイル

次にデータファイルのフォーマットについて見てみましょう。

WAIT 2秒	9C×6回						データ本体	WAIT 1秒
Mark	9C	9C	9C	9C	9C	9C		Mark

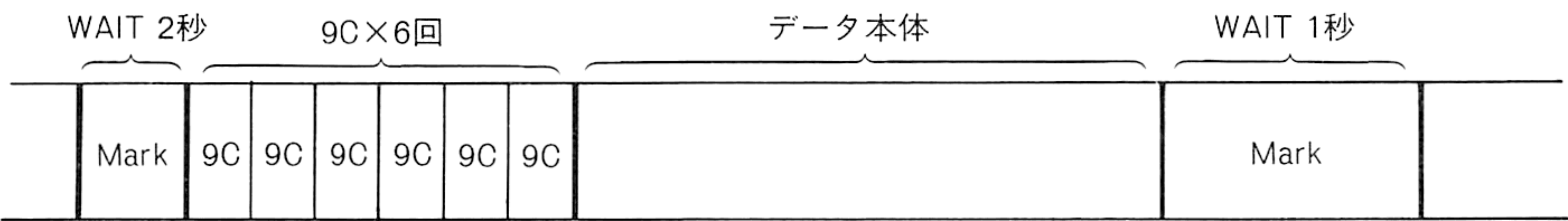



図12-1-B　データファイルフォーマット

最初にテープ走行安定のためのウェイトがあり、9CHを6回書き込んでヘッダを作り、データファイルであることを示します。続いて、データが書き込まれます。データ内容の形式については「12-2　データファイルの書き込み方・読み込み方」で述べます。最後にまたウェイトをして終わります。

一回PRINT＃文、WRITE＃文を実行するたびに、これだけのものがつくられます。したがってデータをこま切れにせず、一回のPRINT＃文で、できるだけ多くのデータを書いた方が処理速度が速くなります。

12-1-3　N88モニタのWコマンド

モニタのWコマンドはチェックサムを付けています。

24H×3回	ファイル名6バイト	WAIT	3AH	アドレスH	アドレスL	チェックサム

チェックサム ← NEG(Hi＋Lo)

3A	データ数	デ　ー　タ	チェックサム
3A	データ数	デ　ー　タ	チェックサム

チェックサム ← NEG（データ数＋データの合計の下位バイト）

3A	00	3A/B9	WAIT

データ数はデータが十分にあるときは80H

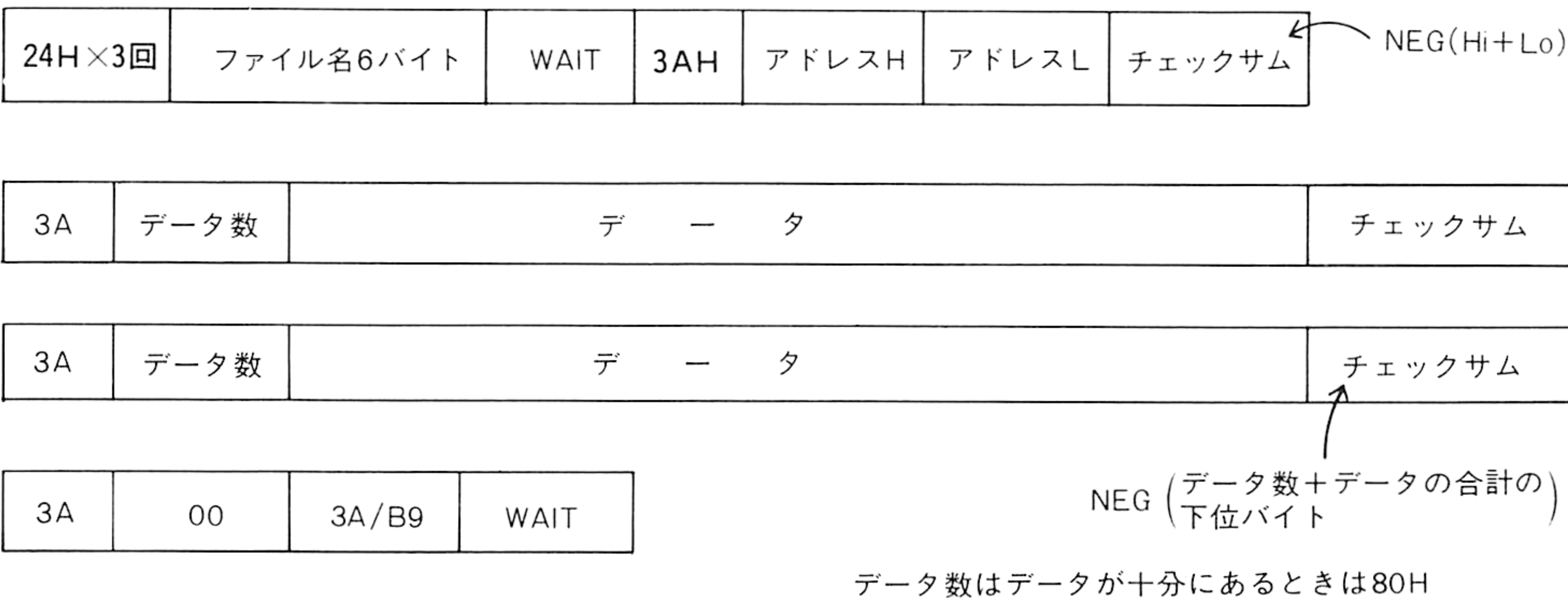



図12-1-C　Wコマンドフォーマット

最初に3バイトの24Hがあり、その後に6バイトのファイル名(6バイトに足りない分は20Hが詰められる)が続き、ファイル名ブロックを形成しています。セーブ時にファイル名を付けないと、このブロックは作られません。0.2秒のWAITのあと3AHがあり、セーブスタートアドレスがあって、次にアドレスのチェックサムがあります。これはスタートアドレスの上位バイトと下位バイトを足したものを1バイトの符号付き2進数とみなし、その符号を逆にしたもの(具体的にはNOTをとって1を足したもの)です。

ここまでがヘッドブロックです。つぎに複数のデータブロックが続きます。(データが少ないときは1ブロックのこともあります。) 各ブロックは3AH、データ数(1バイト)、デー

タ列、チェックサム(1バイト)で構成されます。データ数は普通は128バイトですが、残りデータ数が128バイトないときにはそれ以下になります。チェックサムはデータ数と各データを足し合わせたものの下位1バイトの符号を逆にしたものです。

データを書き終わると、データエンドの印に3A, 00, 3Aまたは3A, 00, B9のパターンを書き込みます。つまりデータ数が0のブロックということで、モニタのRコマンドではデータ数＝0を検出すると、終わりとみなします。

12-2　データファイルの書き込み方・読み込み方

データをカセットに書き込むには

　OPEN"CAS1：ファイル名"　FOR OUTPUT AS ＃n

または

　OPEN"CAS2：ファイル名"　FOR OUTPUT AS ＃n

としてオープンしてから、PRINT＃文、WRITE＃文で実際にデータを書き込みます。(ただし、ファイル名は無視されます。)このときのデータの書き込まれ方はカセット特別で、このことにより種々の現象が起こります。また、一回に書き込むデータの長さが、区切りを含めて255バイトを超えると、読み込めなくなります。

データの書き込まれ方の規則をまとめると、次のようになります。

①PRINT＃文の場合

文字列：　そのまま書き込まれる。

数値：　画面にプリントするときと同じく、文字列に変換して書き込まれる

区切りのコンマ：　そのまま","として書き込まれる。

②WRITE＃文の場合

文字列：　"　"(ダブルクォーツ)で囲まれて書き込まれる。

数値：　画面にプリントするときと同じく文字列に変換して書き込まれる

区切りのコンマ：　そのまま","として書き込まれる。

12-2-1　数値の読み込み

数値も文字列に変換されてから出力されていることに注意してください。つまりカセットに記録されたデータでは数値も文字列であり、読み込む時に文字列として読み込むと、文字列として正常に読み込まれます。

試しに、次のプログラムを走らせてください。

リスト　12-1

```
10 OPEN "cas1:test" FOR OUTPUT AS#1
20 A=20+50
30 B=10^20
40 C$="123"
50 PRINT #1,A,B,C$
60 CLOSE
```

テープに書き込まれるデータのフォーマットは、次のようになります。

ヘッダ	␣70　␣,　␣　1E＋20␣,　123 $^{C}{}_{R}$ $^{L}{}_{F}$	mark

図12-2-A　サンプルデータ1

このファイルを次のプログラムで読むと……

リスト　12-2

```
10 OPEN "cas1:test" FOR INPUT AS#1
20 INPUT #1,A$,B$,C$
30 PRINT A$,B$,C$
40 CLOSE

run
70             1E+20          123
Ok
```

ちゃんと読めます。では読むときに数値変数に読み込む場合、N88-BASICはどうやっているのかというと、実はVAL関数のルーチンを使って、文字列を数値に変換しているのです。

12-2-2　文字列の読み込み

次に、文字列を読み込むときを考えましょう。データとデータの区切りには","（コンマ）が使われています。INPUT＃文で読むときには、これがデータの区切りになります。したがって、文字列の中に","が出てくると、これもデータの区切りとみなされてしまい、データが一部欠けるだけでなく、それ以降のデータがずれてしまいます。

PRINT　#1," ABCDE,FGH"," IJK","LMN"　とすると……

書き込み時：　データ　区切り　データ　区切り　データ

	ABCDE,FGH	,	IJK	,	LMN	$^{C}{}_{R}$	$^{L}{}_{F}$	

読み出し時：　データ　区切り　データ　区切り　データ　区切り　データ

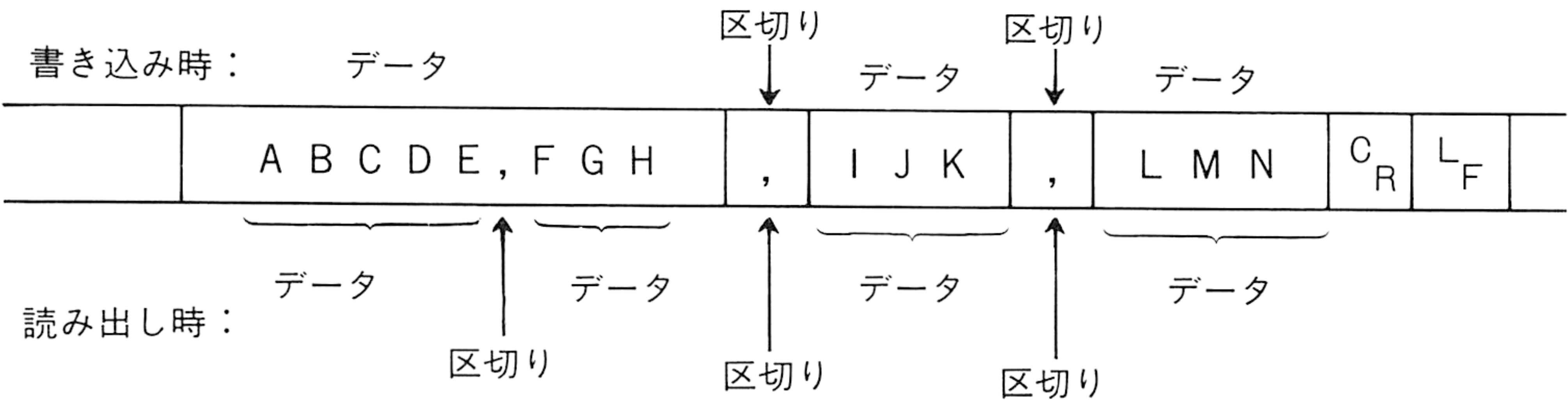



図12-2-B　データのずれ

これを回避するには、文字列を"　"で囲みます。つまりWRITE＃文を使えばよいわけです。しかしディスクがないシステム(model 10)ではWRITE＃文は使えません。このときは文字列の前後にCHR$(34)をつけて

PRINT＃1，CHR$(34)＋"文字列"＋CHR$(34)

というようにします。

ところがこれでも完全ではありません。こんどは文字列内に"が入っていた場合に困ります。カセットから読み込んだ文字列は"で始まっていますから、文字列中に"が現われると、そこで文字列が終わったとみなされます。

結局この問題を完全に解決する方法はありません。なぜならデータ自身もデータの区切りも、どちらも文字であって、同じものだからです。疑似的な解決法としては、各データに対してその長さを書き込んでおく方法があります。つまり書き込み時には

PRINT＃1，LEN(A$)，LEN(B$)，LEN(C$)

PRINT＃1，A$;B$;C$

として、読み込み時には

INPUT＃1，LA，LB，LC

GET＃1，LA＋LB＋LC

FIELD＃1，LA　AS　A$，LB　AS　B$，LC　AS　C$

とします。しかしこの方法では処理が複雑になる上に、文字列の長さを読み書きする分遅くなります。

12-2-3　PRINT＃文の最後のセミコロン

ところでPRINT＃文の最後に；(セミコロン)を付けたらどうなるでしょう。実際にやってみましょう。

リスト12-3　　書き込みプログラム

```
10 OPEN "cas1:test" FOR OUTPUT AS#1
20 A=34 : B$="ABC"
30 PRINT #1,A,B$;
40 CLOSE
```

リスト12-4　　読み込みプログラム

```
10 OPEN "cas1:test" FOR INPUT AS#1
20 INPUT #1,A,B$
30 PRINT A,B$
40 CLOSE
```

実行結果

```
run
Tape read ERROR in 20
Ok
```

Tape read errorが出ましたね。このデータファイルのフォーマットは、画面と同じようにCR(0DH)，LF(0AH)が出力されず、次のようになります。

ヘッダ	␣34　␣，　ABC	mark

図12-2-C　サンプルデータ2

最後のデータをよく見て下さい。何の区切りもなく、いきなり終わっています。このために、データが終わってもなお先を読もうとし、ファイルエンドのmark(＝データがない)に続くSpaceやノイズ等で、リードエラーを起こすためです。従ってこのようにセミコロンを最後につけると、最後のデータのみ読めなくなります。

12-3　N-BASICモードで1200ボーを使う

N88-BASICモードでは、カセットファイルに600ボー(cas 2)と1200ボー(cas 1)の2種の転送速度が使えましたが、N-BASICモードでは600ボーのみしか使用できず、長いデータを作った時など、カセット入出力に時間がかかってしまいます。そこで次のプログラムを実行させると、N-BASICモードで600ボーと1200ボーの両方が使えます。転送速度の切り換えはCMDコマンドを使用します。

CMDB……600ボー

CMDA……1200ボー

このプログラムは、N88-BASICモードのcas 1で作ったデータファイルを読む時にも使用できます。なお、プログラムファイルを1200ボーでCSAVE, CLOADすることもできますが、CLOADの時、このプログラムを実行していなければ全く読めません。

リスト 12-6

```
100 '
110 '---- 1200 bps for N-BASIC mode ----
120 '
130 DEFINT A-Z
140 FOR I=0 TO 58
150   READ A$ : POKE &HF2C5+I,VAL("&H"+A$)
160 NEXT I
170 POKE &HF0FC,&HC3
180 POKE &HF0FD,&HC5
190 POKE &HF0FE,&HF2
200 DATA 7E,D6,42,20,0A,3E,C9,32,B6,F1,32
210 DATA B9,F1,D7,C9,3C,C2,DF,3B,E5,21,EA,F2,22,B7,F1,21
220 DATA F5,F2,22,BA,F1,3E,C3,E1,18,E2,F1,3A,66,EA,E6,0F
230 DATA F6,18,C3,FD,0B,F1,3A,66,EA,E6,0F,F6,1C,C3,50,0C
```

12-4　カセットインタフェース回路の制御方法

カセットの制御ポートとしては、20H, 21H, 30Hがあります。このうち20H, 21HはμPD8251に接続されていて、データのやり取りとそのコントロールに使われます。ポート30Hはシリアルインタフェース回路の状態設定を行ないます。シリアルインタフェース回路はカセットとRS-232Cの両方で共用していますので、RS-232C用にシリアルインタフェース回路(特にμPD8251)を使う場合についてもここで取り扱います。

12-4-1　シリアルインタフェースの状態設定

ポート 30H (OUT)　　N88-BASICのワークエリア：E6C0H番地

7	6	5	4	3	2	1	0
／	／	BS2	BS1	$\overline{\text{MTON}}$	CDS	$\overline{\text{COLOR}}$	$\overline{\text{40}}$

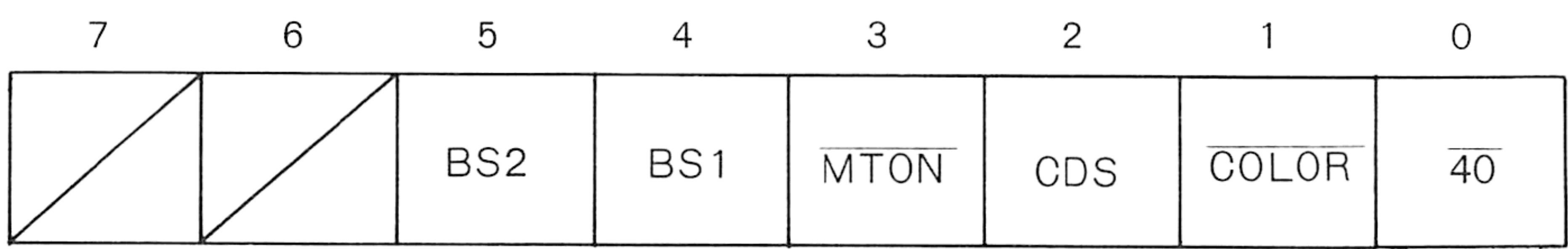



図12-4-A　シリアルインタフェースの設定

CDS	キャリアコントロール
0	マーク
1	スペース

このビットを1にしますと、カセットインタフェースのREC信号はスペースを送り続けますので、カセットは使えなくなります。

MTON	カセットのモータコントロール
0	モータ　オフ
1	モータ　オン

BS2	BS1	シリアルインタフェース切り換え
0	0	カセット　600　ボー
0	1	カセット　1200　ボー
1	0	禁止
1	1	RS—232C

12-4-2　キャリア　ディテクト

RS-232CコネクタのDCDラインの状態を見ます。

ポート　40H（IN）

7	6	5	4	3	2	1	0
／	／	VRTC	CDI	$\overline{\text{EXTON}}$	DCD	$\overline{\text{SHG}}$	BUSY

図12-4-B　DCDライン

DCD	データキャリアディテクト
0	キャリアディテクト　OFF
1	キャリアディテクト　ON

12-4-3　μPD8251の使い方

μPD8251には2つのポートがあり、20Hは実際のデータのやり取りに、21HはμPD8251の制御と状態センスに使われています。μPD8251を使うには、まずイニシャライズを行な

います。イニシャライズ時には、モード設定とコマンド設定を行ないます。

モード設定とコマンド設定は同じ出力ポート(21H)を使用しますので、命令を出力する順番が決まっています。8251をハードウェアあるいはソフトウェアでリセットした後のOUT 21H命令がモード設定で、それ以後のOUT 21H命令はコマンド設定となります。

8251を再度モード設定するためには、ビット6に1をたてたコマンドを設定すると、ソフトウェア的にリセットされ、その後、モード設定、コマンド設定の順に行ないます。したがって、8251を使用する場合、8251にリセットがかかった後の状態(モード設定待ちの状態)であるのか、あるいはモード設定をした後の状態(コマンド設定待ちの状態)であるのかがわかりませんので、2回ダミーコマンドを送るのがよいでしょう。

ダミーコマンドのデータとして05Hを用いれば、8251が上記どちらの状態にあったとしても動作に悪影響は与えません。

8251をキャラクタ長8ビット、ストップビット2ビット、偶数パリティ発生/チェック、ボーレイトを×64モードにイニシャライズした例を示します。

リスト 12-7

```
10 OUT &H21, &H5          :' Send dummy command to 8251
20 OUT &H21, &H40         :' Reset 8251
30 OUT &H21, &HFF         :' Mode instruction
40 OUT &H21, &H5          :' Command instruction
```

N_{88}-BASICおよびN_{88}モニタでのカセット書き込み/読み出し時のμPD8251の設定はこのようになっています。(値は16進)

	モード	コマンド
書き込み	CE	11
読み出し	2E	14

モード及びコマンドの詳細を次に説明します。

(1)モードインストラクションフォーマット

OUT 21H

7	6	5	4	3	2	1	0
S2	S1	EP	PEN	L2	L1	B2	B1

B2	B1	ボーレイト設定
0	1	×1
1	0	×16
1	1	×64

(例)300ボーで転送を行なう場合、8251のクロックTxC、RxCは、

300Hz(×1を使うとき)

4,800Hz(×16を使うとき)

19,200Hz(×64を使うとき)　の周波数に合わせます。

L 2	L 1	キャラクタ長
0	0	5ビット
0	1	6ビット
1	0	7ビット
1	1	8ビット

PEN	パリティ　イネーブル
0	パリティビット　なし/チェックせず
1	パリティビット　発生/チェック

EP	偶数パリティ
0	奇数パリティ発生
1	偶数パリティ発生

S 2	S 1	ストップビット
0	0	無効
0	1	1　ビット
1	0	1/2　ビット
1	1	2　ビット

従ってモードインストラクションとして21HポートにDEHを出力しますとボーレイトはTxC、RxCの1／16、キャラクタ長8ビット、奇数パリティ発生／チェック、ストップビット2ビットというモードになります。

(2)コマンドインストラクションフォーマット

OUT 21H

7	6	5	4	3	2	1	0
0	IR	RTS	ER	SBRK	RxE	DTR	TxEN

TxEN	送信イネーブル
0 1	送信禁止 送信許可

DTR	データターミナルレディ
0	RS—232C　コネクタ上のDTRをロウレベルにします
1	RS—232C　コネクタ上のDTRをハイレベルにします

RxE	受信イネーブル
0 1	受信禁止 受信許可

ER	エラーリセット

1にしますとパリティーエラー、オーバーランエラー、フレーミングエラーのフラグをリセットします。

SBRK	センドブレイクキャラクタ
0 1	通常動作 データ出力を0にします

RTS	送信要求
0	RS—232C　コネクタ上のRTSをロウレベルにします
1	RS—232C　コネクタ上のRTSをハイレベルにします

IR	内部リセット

1にしますと、内部リセットがかかり、μPD8251をモード設定待ちに戻します。

PC-8801mkⅡでは、μPD8251を非同期で用いますので、ビット7は0にして下さい。

(3)データポート

IN／OUT　20H

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0

D0～D7　　　　入出力データ(8ビット)

(4)ステータスの読み出し

IN　21H

7	6	5	4	3	2	1	0
DSR		FE	OE	PE	TxE	RxRDY	TxRDY

TxRDY……送信レディ

このビットが1のとき、μPD8251C内部のデータバスバッファにはデータが入っていないので、CPUから8251にデータを書き込むことができます。このビットはCPUから1キャラクタデータが書き込まれれると、自動的にリセットされます。

RxRDY……受信レディ

このビットが1のとき、8251からCPUに入力すべきデータが8251に入っていることを示します。このビットはデータがCPUに読み出されると自動的にリセットされます。

Tx E……送信エンプティ

8251が送信するデータを持たないとき、このビットは1になります。

PE　……パリティエラー

パリティエラーが検出されると、このビットはセットされます。コマンドインストラクションのERビットでリセットされます。また、パリティエラーが発生しても、8251は動作を継続します。

OE……オーバーランエラー

あるキャラクタを、次のキャラクタを受信し終わる前にCPUが読み出さなかったときに、このビットがセットされます。このビットはコマンドインストラクションのERビットでリセットされます。オーバーランエラーが発生しても8251は動作を継続しますが、オーバーランされた前のキャラクタは消えます。

FE……フレーミングエラー

各々のデータの終わりで有効ストップビットが検出されないときにこのビットはセットされます。このビットもコマンドインストラクションのERビットでリセットされます。フレーミングエラーが発生しても、8251は動作を継続します。

DSR……データ　セット　レディ

RS-232Cコネクタ上のDSR入力の状態を表わします。

12-4-4　プログラム例

(１)１キャラクタ入力するプログラム例です。

```
READY：IN     A,   (21H)……①
       AND  02H………………②
       JR    Z, READY……③
       IN    A,   (21H)……④
       AND  38H…………⑤
       RET  NZ……………⑥
       IN    A,   (20H)……⑦
       RET………………⑧
```

①　ステータスを読み込みます。

②, ③　RxRDYが立つまで、すなわち8251がデータを読み込むまでループします。

④　ステータスを読み込みます。

⑤, ⑥　FE, OE, PEのいずれかのビットが立ちますと、Zフラグ＝0で、もとのルーチンに戻ります。エラーが起きなければ先へすすみます。

⑦　受信データをCPUに読み込みます。

⑧　Zフラグ＝1で戻ります。

エラーが生じた際には、エラーの生じたフラグ(FE, OE, PE)をリセットする必要があります。その例を示しましょう。

```
IN      A, (20H)……①
LD      A, XXH………②
OUT     (21H), A……③
```

①　まず受信バッファの内容を読み出します。これをしないと、受信バッファにデータが残ったままですので、エラーフラグをリセットした直後に8251が次のデータを受信した場合、オーバーランエラーフラグがセットされてしまいます。

②, ③　エラーリセットはビット４に１をたてますが、その他のビットはコマンド設定時と同じにしておきます。コマンド設定時に05Hを使った場合、データを15Hとします。

(２)次に１キャラクタ、出力するプログラム例を示しましょう。

```
       PUSH  AF……………………①
READY：IN      A,   (21H)……②
       AND   01H……………………③
       JR     Z, READY………………④
       POP   AF
       OUT   (20H), A………………⑤
       RET
```

①　送信するデータを待遅させます。

②　ステータスを読みます。

③, ④　8251のバッファが空になり、データをセット可能になるまでループします。

⑤　送信データをセットします。

第13章　RS-232C

RS-232Cとは、元来、通信回線でデータを送受信するモデムとデータ端末装置とを接続するための規格として国際電信電話諮問委員会(CCITT)の勧告を受け、米国のEIA(Electronic Industries Association)が決めた標準的なシリアルデータの伝送規格ですが、現在では、モデムに限らず、シリアルインタフェースとして広く用いられています。PC-8801mk II では、USART (Universal Syncronous ／ Asyncronous Receiver ／ Transmitter) としてμPD8251Cを非周期方式で動作させています。転送速度は、前面のジャンパスイッチにより、18.75ボー～9600ボーまで使用できます。(ただし、BASICでは75ボー～9600ボー までです。)

また、PC-8801mkIIのシリアルインタフェースには、RS-232Cインタフェースとオーディオカセットインタフェースがあり、同じ8251を切り換えて使用しています。カセットインタフェースについては第12章をごらん下さい。

13-1　通信仕様の設定

RS-232Cを使用するには、通信仕様としてモードとボーレイト(データ転送の速度)を決めなくてはなりません。モードの指定はN_{88}-BASICではファイルディスクリプタを使っておこないます。ファイルディスクリプタのデバイス名のところに"COM", ファイル名のところにモードを指定します。したがってRS-232CをN_{88}-BASICで使用する場合にはいわゆるファイル名は存在しません。

モードは通信時のデータ形式や制御情報を指定するもので、パラメータとしてパリティ、データのビット長、ストップビット長、Xパラメータ、Sパラメータ、があります。ターミナルモードではさらに全二重／半二重、スタック長が指定できますが、ここではN_{88}-BASICで行なう場合のみを考えます。さて、N_{88}-BASICでは

```
OPEN  " COM：①②③④⑤"  FOR……
```

のように5文字で指定します。

①パリティ　　E：偶数パリティチェック　(Even)
　　　　　　　O：奇数パリティチェック　(Odd)
　　　　　　　N：パリティチェックなし　(None)

②データのビット長　　7：7ビット
　　　　　　　　　　8：8ビット

③ストップビット長　　1：1ビット
　　　　　　　　　　2：1.5ビット
　　　　　　　　　　3：2ビット

④X パラメータ　　　X：バッファオーバーフローコントロールを行なう
　　　　　　　　　　N：バッファオーバーフローコントロールを行なわない

⑤S パラメータ　　　S：シフトコード制御を行なう
　　　　　　　　　　N：シフトコード制御を行なわない

たとえば、偶数パリティ、データ長8ビット、ストップビット1ビット、Xパラメータ有効、Sパラメータ有効とすると、

　　”COM：E81XS”

となります。この章ではこのモードを使用します。

ボーレイトはPC-8801mkⅡの前面のジャンパースイッチで指定します。ファイルディスクリプタで変更することはできません。

RS-232C
1 2 3 4 5 6 7 8
9600ボー
4800ボー
2400ボー
1200ボー
600ボー
300ボー
150ボー
75ボー

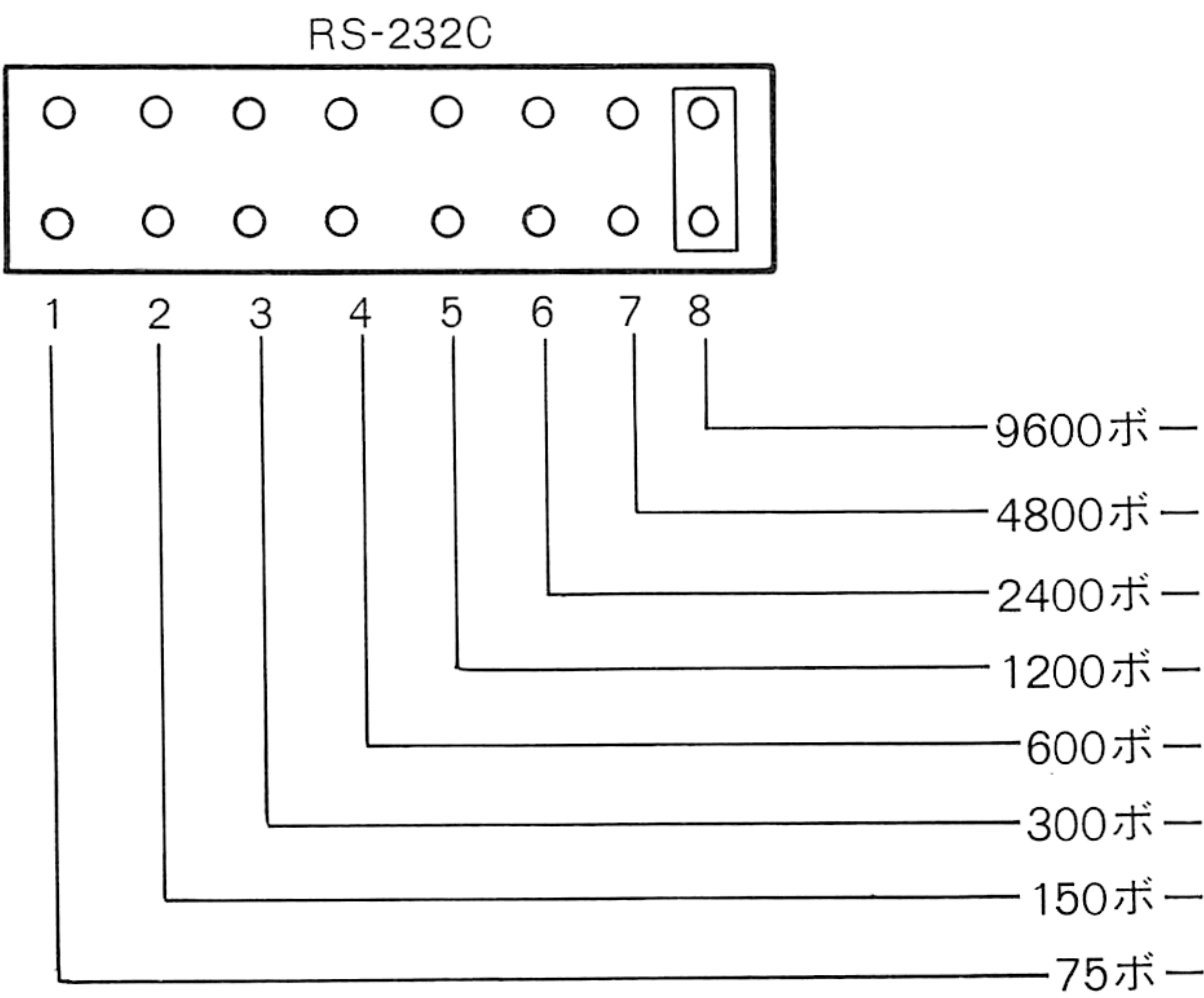



図　ジャンパスイッチとボーレイト

13-2　2台のPC-8801mkⅡを接続する

「PC-8801mkⅡと他のコンピュータとの間でデータのやり取りを行ないたい」、といった事はよくあります。

例えば、会計、在庫管理システムにおいて、売り上げや出納、入出庫などのマスターファイルをホストコンピュータ上で作成、そのホストに多数のPCを接続し、各部課からのデータをPCに入力、ホストのファイルに登録する。あるいは、計算の一部をホストが行ない、その間PCは他の計算を行なう。そして結果をホストからもらい、PCの結果と合わせ最終結果を出力する、等々、数多くの応用が考えられます。この様な時、ホスト側にRS-232Cポートがあれば、簡単にPCと接続できます。

ここではその基本的な例として、2台のPC-8801mkⅡを接続して、データやプログラムを転送することを考えます。

13-2-1　接続用専用ケーブル

RS-232Cはもともとコンピュータ(データ端末装置)とモデムを接続するためのものでしたから、コンピュータ同士を接続するにはRS-232Cケーブルの信号線を一部変更する必要があります。これは自分で作成してもよいのですが、PC同士を接続するにはすでに変更してあるケーブルがNECから出ているので、それを利用するのがよいでしょう。型番はPC-CA602で、品名はRS-232C用ケーブル(リバース)といいます。

自分で作る場合は次のようにします。まず、RS-232C用オスコネクタを2つ用意します。次に、配線図の様にRxDとTxD、DSRとDTR、といった対になる信号線を入れ換えて接続するのです。ただPCの場合、8番ピンのDCD(キャリア検出信号)は、対になるピンがなく、GND(7番ピン)に接続しておきます。これで、DCDは常にアクティブとなります。これで、PC同士が接ながります。対になる信号を入れ換えたことで、各PCからは、他方が等価的にモデムのように見えるためです。

もしPC以外のコンピュータと接続するケーブルを作るときには、PCにないピンが使用されていたり、逆にPCにあるピンが使用されていなかったり、あるいは、先のDCDに＋5～＋15Vの電圧をかける必要があるものもありますので、十分確認をして下さい。

GND 1 — GND 1
TXD 2 — RXD 3
RXD 3 — TXD 2
RTS 4 — CTS 5
CTS 5 — RTS 4
DSR 6 — DTR 20
DTR 20 — DSR 6
GND 7 — GND 7
DCD 8 — GND 7 / DCD 8 — GND 7

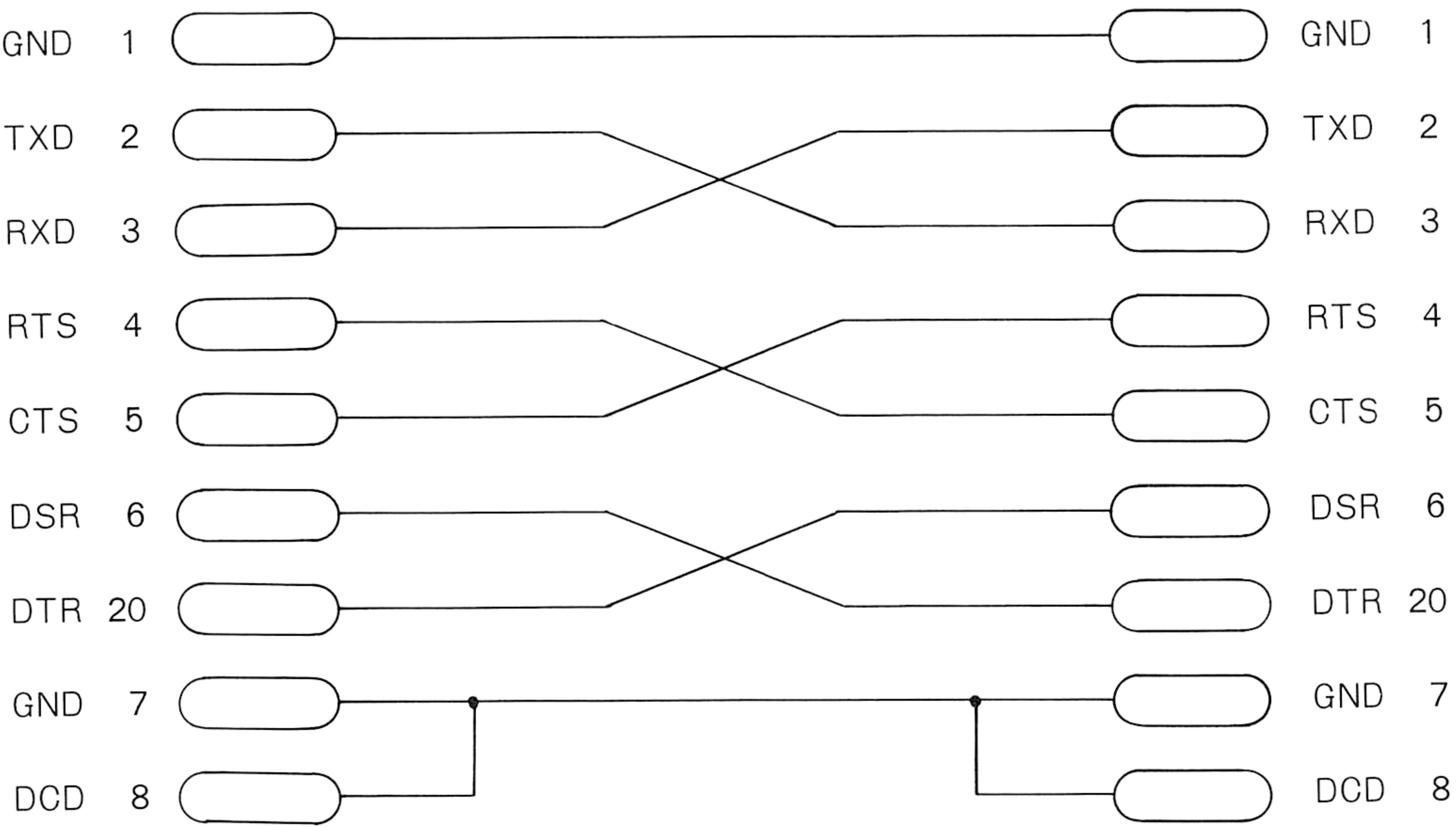



図13－2－A　信号の入れ換え

端子番号	信号名	端子番号	信号名	ピンコネクション
1	GND	14	NC	13 - - - 1
2	$\overline{\text{TXD}}$	15	NC	
3	$\overline{\text{RXD}}$	16	NC	
4	RTS	17	NC	
5	CTS	18	NC	
6	DSR	19	NC	
7	GND	20	DTR	
8	DCD	21	NC	
9	NC	22	NC	
10	NC	23	NC	
11	NC	24	NC	
12	NC	25	NC	25 - - - 14
13	NC			

信号名	ピン番号	機　　能	信号名	ピン番号	機　　能
GND	1,7	1…保安用アース 7…信号用アース	DSR	6	モデムからの動作可能信号
$\overline{\text{TXD}}$	2	送信データ	DCD	8	キャリア検出
$\overline{\text{RXD}}$	3	受信データ			
RTS	4	モデムへの送信要求	DTR	20	ターミナルの動作可能信号
CTS	5	モデムからの送信可能信号			

図13－2－B　RS-232Cコネクタ

13-2-2 データの転送

では、2台のPC-8801mkⅡでデータを転送してみます。この時、双方のボーレイトを合わせるのを、忘れないで下さい。(600ボーがよいでしょう)

RS-232Cにデータを出力するには、他のシーケンシャルファイルと同様、次のように行ないます。

送信側

```
10 OPEN "com:E81XS" FOR OUTPUT AS#1
20 A$="ABCDE,FGHIJ"
30 B$="KLMNO"
40 WRITE #1,A$,B$
50 CLOSE
```

PRINT#文ではなく、WRITE#文を使っていることに注意してください。これは、データ間に区切りとして、コンマ(,)を出力し、文字列は"　"で囲まれて出力されるからです。この時のデータフォーマットは次のようになります。

```
"ABCDE,FGHIJ","KLMNO" CR LF
```

図13-2-C 送信フォーマット

これを次のプログラムで読むことができます。

受信側

```
10 OPEN "com:E81XS" FOR INPUT AS#1
20 INPUT #1,A$,B$
30 PRINT A$,B$
40 CLOSE
Ok


run
ABCDE,FGHIJ    KLMNO
Ok
```

さて、このように基本的にはディスクのシーケンシャルファイルと同様なのですが、1つだけそれと異なって注意しなければならない事があります。それは、バッファの限界です。

RS-232Cの入力は、内部的にはインタラプトで処理されており、データを1文字受けるたびにメインRAM上のファイルバッファに蓄え、INPUT#文でこのバッファからデータを取り出すわけです。従って、INPUT#文によるデータの取り出し速度より、受信したデータを蓄積していく速度の方が速ければ、バッファ内のデータはどんどん増え、ついにはバッファからあふれてしまいBuffer overflowとなってしまいます。特に、ボーレイトが速くデータの量が多い時は注意しなければなりません。

この現象を防ぐには、アクノリッジを返す方法があります。つまり、データを送ったら、相手が受け取った事を示す返事を出すまで次の送出を待っているのです。ただし、アクノリッ

ジの送出を待っているため、その分出力側の処理速度が落ちてしまいます。その例を示します。

送信側

```
10 A$="ABCDE,FGHIJ"
20 OPEN "com:E81XS" AS#1
30 FOR I=0 TO 5
40   WRITE #1,I,A$       :' out data
50   INPUT #1,AC$        :' in acknowledge
60 NEXT
70 CLOSE
```

受信側

```
10 OPEN "com:E81XS" AS#1
20 FOR I=0 TO 5
30   LINE INPUT #1,A$    :' in data
40   PRINT #1,CHR$(4)    :' out acknowledge
50   PRINT A$
60 NEXT
70 CLOSE
```

これは割り込みを使うと改良されます。割り込みを使用すると反応が速く、送受双方の処理速度が向上します。割り込みについての説明は第14章を参照してください。

次に示すプログラムは、割り込み処理でアクノリッジを返すもので、処理速度向上のため、受信データをキューバッファに入れ、メインルーチンでは、このキューバッファから読み込むようにしています。受信側のプログラムをこのプログラムと入れ換えてください。

受信側

```
100 '
110 '   RS-232C with Acknowledge
120 '
130 N=10:GOTO 220
140 '-------- Interrupt routine --------
150  COM OFF
160  LINE INPUT #1,A$(WSP)                          ' Put to queue
170  WSP=(WSP+1) MOD N
180  IF ((WSP+1) MOD N)=RSP THEN FULL=-1:RETURN ELSE FULL=0
190  PRINT #1,CHR$(4)
200  COM ON
210 RETURN
220 '-------- MAIN --------
230 OPEN "com:E81XS" AS #1
240 ON COM GOSUB 140 : COM ON
250 '
260 IF RSP=WSP MOD N THEN 300
270   A$=A$(RSP):RSP=(RSP+1) MOD N                  ' Get from queue
280   IF FULL THEN GOSUB 180
290   PRINT A$
300 FOR I=0 TO 100: PRINT ".."; :NEXT I             ' Dumy loop
310 PRINT
320 GOTO 260
```

13-2-3　プログラムの転送

次はプログラムを転送してみましょう。メモリからメモリへ転送する場合とディスクからディスクへ転送する場合を考えてみます。

(1)メモリからメモリへ

プログラムをダイレクトモードで転送することは簡単で、送信側でSAVE、受信側でLOADを行なえば良いのです。ただ、この時受信側では、受信が完了してもLOADコマンドから抜け出さず、送信側のSAVEが終わってからSTOPキーを押さなくてはなりません。この理由を説明しましょう。

転送時のフォーマットは、ディスクにアスキーセーブする時と同じでプログラムがアスキーコード列として送られます。ところが、ディスクのアスキー形式ファイルと違うのは、プログラムの最後を示すエンドマークがないことです。

LOADの時にはRS-232Cからの入力に対して、キー入力と全く同様の動作でプログラムを格納していきます。この時、別にエンドマークのたぐいは識別せず、また仮に識別していても、送信側で送らないのですから、いつまでたってもLOADコマンドから抜け出せません。したがって、人間が受信側のSTOPキーを押さなくてはならなくなるわけです。(これでも、プログラムは正常に入るはずです)

送信側と受信側が近くにあればこれでもよいのですが、送信側と受信側が離れている時など、SAVEコマンドの終了を確認できない時どうすればよいでしょう。転送にかかる時間をあらかじめ調べておき、ころ合いを見てSTOPキーを押すのも1つの方法です。しかし、もっとよい方法があります。送信側で、次に示すようにダイレクトモードで実行させるのです。

```
save "com:E81XS"
Ok
open "com:E81XS" for output as#1
Ok
print #1,"beep"
Ok
close
Ok
```

受信側では単にLOADコマンドのみで結構です。

```
load "com:E81XS"
Direct statement in file
Ok
```

LOADコマンドからエラーで抜け出していますが、プログラムは正常に入っています。なぜエラーとなるか分かると思いますが、送信側でプログラム・ファイルを送った後、ファイルをOPENして"beep"と送っています。これは、実は行番号をつけていなければ何でもよいのですが、受信側ではこれをダイレクトステートメントと解し、Direct statement in fileとなったわけです。これで、何とかプログラムを送ることができました。

ではダイレクトモードでのSAVEとしてプログラムを送るのでなく、BASICの管理下でプログラムの送出ができないでしょうか？これが行なえるのです。次のプログラムを送信側

で走らせ、受信側でLOADを実行させます。すると、

送信側

```
10 OPEN "com:E81XS" FOR OUTPUT AS#1
20 PRINT #1, "10 for i=0 to 100"
30 PRINT #1, "20   beep 1 : beep 0"
40 PRINT #1, "30 next i"
50 PRINT #1, CHR$(4)
60 CLOSE
```

受信側

```
load "com:E81XS"
Direct statement in file
Ok
list
10 FOR I=0 TO 100
20   BEEP 1 : BEEP 0
30 NEXT I
Ok
```

ちゃんと入ったでしょう。今まで、データファイルとプログラム・ファイルを別のものとして考えて来ましたが、その差は受け取る側がプログラムと考えるかデータと考えるかの違いのみで、本質的には同じものなのです。

(2)ディスクからディスクへ

こんどはディスク上のプログラムを送ることを考えましょう。次のプログラムは、ディスク上のアスキー形式プログラムファイルをRS-232Cに送出し、受信側では受けとったプログラムをディスク上にアスキー形式プログラムファイルを作成するものです。

送信側

```
100 FILES : PRINT
110 INPUT "Type in file name  ";NA$
120 IF LEN(NA$)>9 THEN PRINT " Too long " : GOTO 110
130 OPEN "1:"+NA$ FOR INPUT AS#1
140 OPEN "com:E81XS" FOR OUTPUT AS#2
150 WHILE NOT EOF(1)
160   LINE INPUT #1,A$
170   PRINT #2,A$
180 WEND
190 PRINT #2,CHR$(4)
200 CLOSE
```

受信側

```
100 FILES : PRINT
110 INPUT "Type in file name  ";NA$
120 IF LEN(NA$)>9 THEN PRINT " Too long " : GOTO 110
130 OPEN "com:E81XS" FOR INPUT AS#1
140 OPEN "1:"+NA$ FOR OUTPUT AS#2
150 '
160   LINE INPUT #1,A$
170   IF A$=CHR$(4) THEN 200
180   PRINT #2,A$
190 GOTO 160
200 CLOSE
```

13-3　機械語によるRS-232Cの制御

機械語によってPC-8801mkⅡのRS-232C機能を制御するのに便利なROM内サブルーチンを説明しましょう。これはBASICのRS-232Cを制御する文に相当する機械語のサブルーチンで、これらを利用すると機械語でも手軽にRS-232Cを制御できます。

① COM 1 のオープン

アドレス： 7BC2H

ENTRY：

[EC8F～]　モード設定コード
HL　コミュニケーションバッファのアドレス
DE　0FE04Hにしておきます。
[E6ED] と [E6E8] は、0にしておきます。

注意： このルーチンは、リターンする時スタックから4バイト余分にとりますので、コールする前にダミーを入れておく必要があります。

(例) OPEN "COM：E81XS" をこのルーチンを使って書くと次のようになります。

```
FNAME       EQU    0EC8FH
OPEN_COM1   EQU    7BC2H
            LD     HL, COMBUF
            CALL   OPEN_COM
            .
            .
            .
            RET

OPEN_COM:   PUSH   HL
            PUSH   AF          ;ダミーの4バイトをスタックへ
            XOR    A
```

```
                LD      (0E6E8H), A
                LD      (0E6EDH), A
                LD      DE, 0FE04H
                JP      OPEN_COM1
COMBUF:         DB      10+256      ;コミュニケーションバッファ

                ORG     FNAME
                DB      " E81XS", 0
                .
                .
                END
```

② COM1への1文字出力

アドレス： 3226H

ENTRY： A キャラクタコード

③ COM1へのカナ文字出力

COM1にSI／SOプロトコルでカナ文字を出力します。

アドレス： 3203H

ENTRY： A カナ文字コード

④ COM1のポーリング

アドレス：7570H

EXIT ：HL 受信した文字数

フラグ等は変化します。

⑤ COM1からの1文字入力

アドレス：7C9BH

ENTRY：A 1を入れます。（キューナンバー1を表わします。）

EXIT ：A キャラクタコード

⑥ COM1のクローズ

アドレス：7C57H

ENTRY：〔スタックのトップ〕ファイルポインタ VARPTR(#n)

EXIT ：〔E6ED〕 0が入ります。

このルーチンはリターンするとき、ファイルポインタの他にスタックから2バイト余分にとりますので、コールする前にダミーを入れておく必要があります。

（例）

```
CLOSE  COM1   EQU   7C57H
              LD    HL, COMBUF
              CALL  CALL  CLOSE
              ・
              ・
              ・
              RET
CLOSE COM：   PUSH  HL          ；ダミーの２バイトをスタックに入れます
              PUSH  HL          ；ファイルポインタをスタックに入れます
              J P   CLOSE COM1
              ・
              ・
              END
```

第14章　割り込み

N₈₈-BASICではファンクションキーやSTOPキーなどを押した時にあらかじめ指定した行へGOSUBさせる機能があります。これを便宜上BASIC割り込みと呼ぶことにします。また、機械語レベルではソフトウェアでマスク可能な8レベルの優先順位をもつ割り込み機能を持っています。これを機械語割り込みと呼ぶことにします。機械語割り込みはハードウェアと密接な関係を持っていますから、ある程度ハードウェアの知識がないと使いこなすことはできません。

14-1　BASIC割り込み

14-1-1　BASIC割り込みステートメントの機能

ここではHELPキーを使ったBASIC割り込みを例にとって説明します。プログラムの最初にON HELP GOSUB×××で割り込み処理ルーチンを定義し、割り込みを可能にするHELP ON命令を実行すれば、HELPキーを押した時に、定義した処理ルーチンにGOSUBします。この割り込み関係のステートメントを示します。

ON HELP GOSUB	割り込み処理ルーチンの定義
HELP ON	割り込み可能
HELP OFF	割り込み禁止
HELP STOP	割り込み一時保留

この中で、HELP OFFとHELP STOPの違いは明確にしておいて下さい。HELP OFFは割り込みそのものを無視するもので、HELP OFFの間にキーが押されても割り込みはおこりません。一方、HELP STOPの方では、割り込みは発生しますが、これを受けつけ、実際の処理ルーチンへ行くのを保留しておくのです。ですから、HELP STOPを解除した段階(HELP ON)で受けつけられ、処理ルーチンへ制御を移します。この時、HELP OFFを実行すれば保留されていた割り込みはキャンセルされます。

モード	HELP ON	HELP STOP	HELP ON
割り込み		割り込み発生	実際に処理ルーチンへ
受付状態		保留	受けつけ

モード	HELP ON	HELP STOP	HELP OFF	HELP ON
割り込み		割り込み発生		処理ルーチンは実行されない
受付状態		保留	キャンセル	

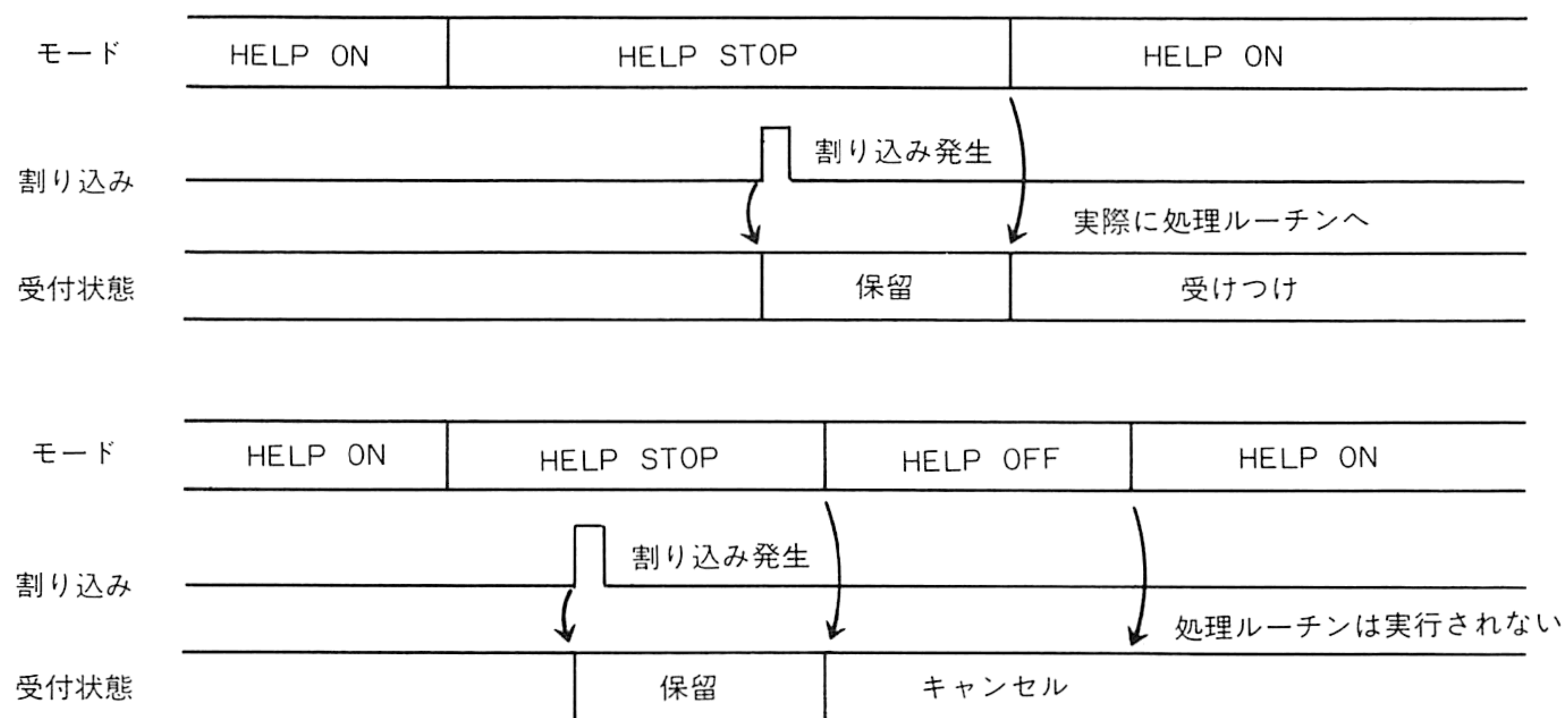



図14－1－A HELP ON／OFF／STOP

HELP STOPを解除し割り込み可にするのはHELP ONです。また、割り込み保留状態にするには、一旦HELP ONにし、その後 HELP STOP にしなければなりません。図に示すと次の様になります。

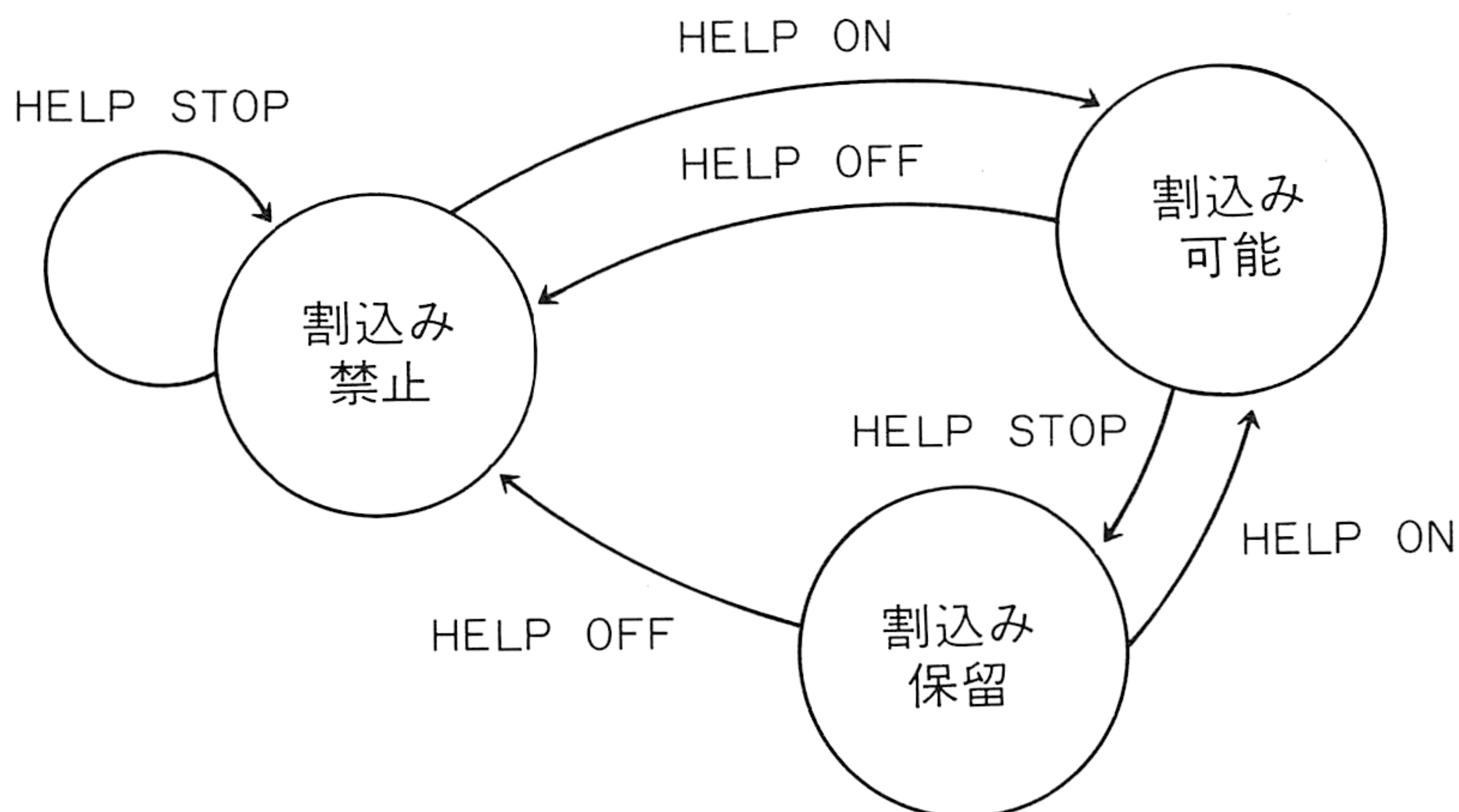



図　HELP割り込みモードの変化

14-1-2　BASIC割り込みはどこでかかるか

いままでは単にHELP ONのときHELPキーを押すと割り込みがかかると考えてきましたが、はたして押した瞬間にかかるのでしょうか。このプログラムを実行して、PAINTを行なっている間にHELPキーを押してみてください。

```
10 SCREEN 0,0 : ROLL 197
20 ON HELP GOSUB 70 : HELP ON
30 LINE (100,100)-(200,180),5,B
40 PAINT (150,150),6,5
50 HELP OFF
60 END
70 PRINT "HELP INTERRUPT!!"
80 HELP OFF
90 END
```

PAINTが終わってからHELP INTERRUPT！と表示されましたね。つまりBASIC割り込みは文の実行中に割り込み要因が起こっても、文の実行が終わるまでは保留になっているわけです。別の言い方をすれば、BASIC割り込みは文と文の間でかかるわけです。

ただし、INPUT文は例外です。これは入力待ちのときにもかかります。(ファンクションキー割り込みがかからないのはわざとそうしているのです。)また、入力待ちのときにかかった割り込みからただのRETURNで戻ると、マニュアルとは異なって、INPUT文がある行の次の行から実行を再開することに注意してください。INPUT文から後の文は無視されるわけです。

14-2　機械語割り込み

機械語割り込みはハードウェアからの要求によって生じ､割り込みルーチンへの分岐もハードウェアによって自動的に起こります。Z-80(μPD780)の割り込みにはノンマスカブルインタラプト(NMI)とマスカブルインタラプトがあります。NMIはソフトウェアによって禁止できない割り込みで、スロットバス上にピンが出ていますが、ソフトウェアでのサポートがないため使用できません。ただし、64KRAMモードにして、独自にソフトウェア／ハードウェアを作成すれば使用することもできます。これに対してマスカブルインタラプトはソフトウェア(EI／DI命令)で許可／禁止の制御ができるものです。また、PC-8801mkⅡではいくつかの割り込みを個々に許可／禁止することもできます。

14-2-1　割り込みテーブル

N88-BASIC、N-BASICではZ-80(μPD780)は、インタラプトモード2で動いています。このモードでは割り込みルーチンのアドレスは､割り込みテーブルを参照して決められます。つまり、割り込みの種類によってあらかじめ指定されたアドレスへ飛んで行くわけで、BASICのON KEY GOSUB ××××，××××，…と考え方としては似たようなものです。この××××，××××，…に相当するものが割り込みテーブルです。PC-8801/mkⅡでは8レベルの割り込みが可能ですから､割り込みテーブルも8つのアドレスからなっています。

(1)　N-BASIC動作時の割り込み

N-BASIC動作時は割り込み機能を使用していないため、8レベルの割り込みはすべて使用することができます。ただし、優先順位が高いほうの3チャネル(RXRDY、VRTC、CLOCK)は、その用途に合わせたハードウェアと接続されているので、他の目的に使用することができません。また$\overline{\text{FDINT}}$ 1、$\overline{\text{FDINT}}$ 2割り込みは、通常のユーザ割り込みと同様に使用することができます。

優先順位	アドレス	アドレステーブルの内容　（初期値）	チャネル	用　　　途	BASICでの使用	レベル
高	8000 1	E6 F1	RXRDY	CMT RS-232C 受信割り込み	NO	0
↑	8002 3	E9 F1	VRTC	画面終了割り込み	NO	1
	8004 5	A5 7F	CLOCK	リアルタイムクロック(1/600s)	NO	2
	8006 7	A5 7F	INT 4	ユーザ割り込み	NO	3
	8008 9	7F 23	INT 3	〃	NO	4
	800A B	A9 23	INT 2	〃	NO	5
↓	800C D	A5 7F	FDINT1	〃	NO	6
低	800E F	A5 7F	FDINT2	〃	NO	7

N-BASICの割り込みテーブル表

(2)　N88-BASIC動作時の割り込み

N88-BASIC動作時は、RXRDY、VRTC、CLOCKの各割り込みを使用しています。さらに8インチのフロッピーディスクを使用する場合は、FDINT 1、FDINT 2割り込みも使用しますのでこれらのチャネルの割り込みテーブルを安易に書き換えると暴走します。つまり、INT 4，INT 3，INT 2の3チャネルがユーザに開放されているわけです。

優先順位	アドレス	アドレステーブルの内容	チャネル	用　　　途	BASICでの使用	レベル
高	F300 1	EA E7	RXRDY	CMT RS-232C 受信割り込み	YES	0
↑	F302 3	08 E8	VRTC	画面終了割り込み	YES	1
	F304 5	0F E8	CLOCK	リアルタイムクロック(1/600s)	YES	2
	F306 7	14 E8	INT 4	ユーザ割り込み	NO	3
	F308 9	14 E8	INT 3	〃	NO	4
	F30A B	14 E8	INT 2	〃	NO	5
↓	F30C D	1A E8	FDINT1	FDD用リザーブ	YES	6
低	F30E F	20 E8	FDINT2	FDD用リザーブ	YES	6

N88-BASICの割り込みテーブル表

14-2-2　割り込みチャネル

ユーザ割り込みをかけるためにはスロットバス上の対応するピンに割り込み要求信号を入れます。割り込み制御回路には割り込み要求を保持するフリップフロップがありますので、スロットバス上のチャネルにネガティブエッジの要求信号を入力することにより割り込みを発生させることができます。したがって、割り込み要求信号は、割り込みが受け付けられるまでロウレベルに保つ必要はありません。またこのフリップフロップは、そのレベルの割り込みが受け付けられると自動的にリセットされ、割り込み要求は解除されます。

14-2-3　割り込みコントロールポート

割り込みルーチンの説明のために割り込みコントロールポートの働きを理解しておく必要があります。PC-8801mkⅡでは割り込みコントロール用のポートとしてポートE4H，E6H

があります。

① ポート　E4H

このポートによって、割り込みコントロール回路のカレントステータスレジスタに優先度判断のための現在のインタラプトレベルと、優先度判断方法のモードを書き込みます。

ポートE4H (OUT)　N_{88}-BASICのワークエリア：E6C3H番地

7	6	5	4	3	2	1	0
／	／	／	／	$\overline{SGS}$	$\overline{B_2}$	$\overline{B_1}$	$\overline{B_0}$

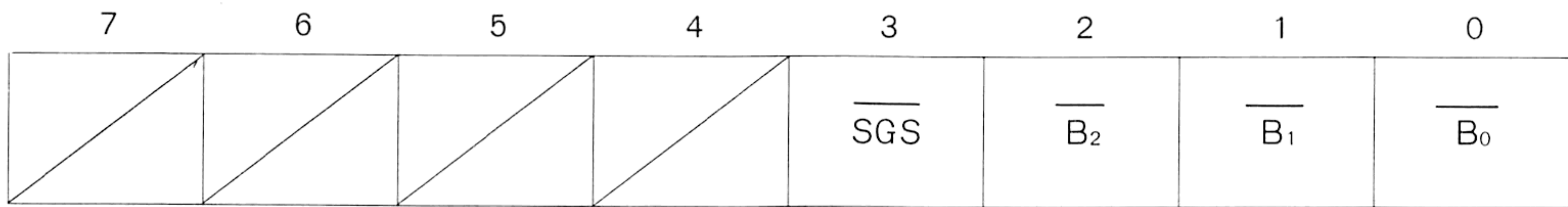



図14－2－A　ポートE4H

$\overline{B2}$　$\overline{B1}$　$\overline{B0}$　　インタラプトレベルの設定(０～７：０が最高レベル)

$\overline{SGS}$　　カレントステータスレジスタのコントロール
　０　　インタラプトレベルと比較し割り込み発生
　１　　優先順位のみによる割り込み発生

通常は$\overline{SGS}$＝０で使用します。インタラプトレベルの設定をするときにはN_{88}-BASICのときE6C3H番地、N-BASICのときEA55H番地に同じものをストアしておきます。これはインタラプトのネスティングが起こったときのために、以前のインタラプトレベルを知っておく必要があるからです。

② 割り込みマスクフラグ

このポートはRXRDY, VRTC, CLOCKの３種の割り込みの許可／禁止を個々に設定するものです。

ポートE6H (OUT)　N_{88}-BASICのワークエリア：EF0EH番地

7	6	5	4	3	2	1	0
／	／	／	／	／	$\overline{RXMF}$	$\overline{VRMF}$	$\overline{RTMF}$

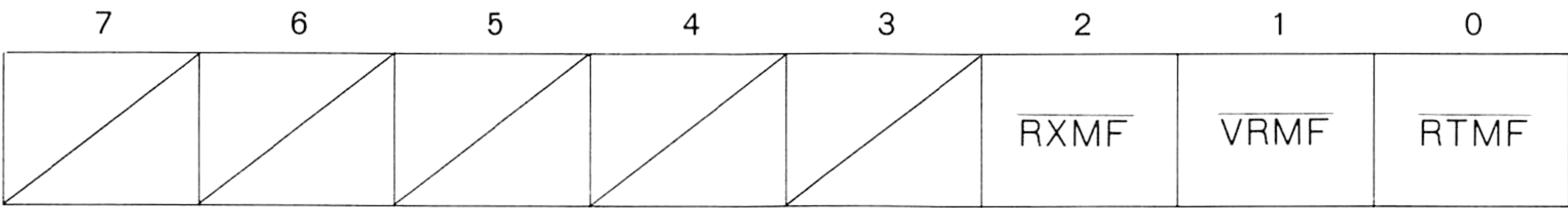



図12－2－B　割り込みマスクフラグ

$\overline{RTMF}$　　CLOCK割り込みマスククラブ
　０　：　割り込み禁止
　１　：　割り込み許可
$\overline{VRMF}$　　VRTC割り込みマスククラブ
　０　：　割り込み禁止
　１　：　割り込み許可
$\overline{RXMF}$　　RS-232C受信割り込みマスククラブ
　０　：　割り込み禁止
　１　：　割り込み許可

14-2-4　一般的な割り込みの使い方

N88-BASICでのユーザ割り込みを用いた標準的な例を示します。この例では割り込み処理ルーチンはメインRAM上(8400H～FFFFH)に置く必要があります。割り込みを使うためには、まず、割り込みコントロール回路のカレントステータスレジスタを操作する必要があります。

(1)初期設定

```
DI
LD     A, 0F3H
LD     I, A        ;Iレジスタに割り込みテーブルの上位アドレスをセット
LD     A, 07H
OUT    (0E4H), A
LD     HL, INT     ;インタラプト処理ルーチンアドレス
LD     (その割り込みのテーブルアドレス), HL
EI
```

割り込みコントロール回路のカレントステータスレジスタに全てのレベルの割り込みを許可し、割り込みテーブルに割り込み処理のアドレスを設定します。ここでラベルINTが割り込み処理ルーチンの入口を表わしています。

(2)割り込み処理

```
INT:    DI
        PUSH   AF
        PUSH   HL
        PUSH   DE                      ①
        PUSH   BC
        LD     A, (0E6C3H)
        PUSH   AF                      ②
           :
           :                           ③
           :
        LD      A, 割り込みレベル
        OUT     (0E4H), A              ④
        LD      (0E6C3H), A
        EI
           :                           ⑤
           :
        POP     AF
        LD      (0E6C3H), A            ⑥
        OUT     (0E4H), A
```

```
        POP     BC
        POP     DE
        POP     HL                      ⑦
        POP     AF
        RET
```

① レジスタを退避します。

② E6C3H番地には、E4Hポートに出力したデータが入っていますので割り込み直前の割り込みレベルが保存されています。これは、この手続きで変更されてしまいますので、スタックに退避します。

③ DI状態での割り込み処理を行ないます。

④ 割り込みレベルを割り込みコントロール回路のカレントステータスレジスタに書き込みE6C3H番地にも保存します。

⑤ EI状態で実行させることが可能な処理を行ないます。

⑥ ②で保存した値を復帰させ、割り込み発生直前の割り込みレベルをカレントステータスレジスタに復帰させます。

⑦ 全てのレジスタを復帰させ、割り込まれたプログラムに戻ります。

N-BASICのときには割り込みテーブルのアドレスを変え、E6C3H番地の代わりにEA55H番地を使うだけです。

14-2-5 N88-BASICでの割り込み処理

「14-2-1割り込みテ-ブル」にあるように、N88-BASICでは割り込みルーチンのエントリはRAM上に置かれています。ここには、割込みレベルを割込みコントローラにセットし、メモリバンクをN88-BASIC ROMにもどして、本当の割込み処理ルーチンをCALLするプログラムがすでに書き込まれています。これはRAM上にあるので書き換えることも可能ですが、やはりBASICインタプリタが使用しているので十分な注意が必要です。

N88-BASICでの割込み処理ルーチンは以下のとおりです。

●RXRDY

テーブルアドレス：F300H

割り込みルーチンエントリ：E7EAH

ROM内割り込み処理ルーチン：3167H

シリアルインターフェースから受取ったデータを入力バッファに入れます。RS-232Cの場合はXパラメータの処理も行います。

●VRTC

テーブルアドレス：F302H

割込みルーチンエントリ：E808H

ROM内割込み処理ルーチン：3080H

カーソルのON／OFF、ライトペン入力処理、キー入力処理を行います。

●CLOCK
テーブルアドレス：F304H
割込みルーチンエントリ：E80EH
ROM内割込み処理ルーチン：4143H
INPUT WAITの時間処理、ON TIME$ GOSUBの時間処理、FDDの時間処理を行います。

●FDINT 1
テーブルアドレス：F30CH
割込みルーチンエントリ：E81AH
ROM内割込み処理ルーチン：3CB9H
5インチDMAタイプの動作終了・状態変化割り込みの処理を行います。

●FDINT 2
テーブルアドレス：F30FH
割込みルーチンエントリ：E820H
ROM内割込み処理ルーチン：3CACH
8インチDMAタイプの動作終了・状態変化割り込みの処理を行います。

14-2-6　VRTC割り込み

VRTC割り込みは一画面の表示が終わるたびにかかる割り込みで、標準CRTのときは約16msごと、高解像度CRTのときは約17msごとにかかります。さきほどN88-BASICではVRTC割り込みが既に使用されていてユーザは使用できないと述べましたが、VRTC割り込みはとても重宝な機能なのでぜひとも使いたい機能です。そこで、ここではVRTC割り込みを使用する方法を説明しましょう。もちろんN88-BASICの機能は止めません。また、クロック割り込み(1.67msごと)も同様の方法で使用できるようになります。

N88-BASICではVRTC割り込みのルーチンエントリはE808Hです。ここには

```
E808:    PUSH  HL
E809:    LD    HL, 3080H
E80C:    JR    0E7EEH
```

というルーチンが置かれています。この3080HというのがVRTC割り込みの核となる処理ルーチンのアドレスです。ですから、ここを自分で作ったルーチンのアドレスに書き換えて、そのルーチンの終わりで3080Hに飛ばしてやればよいわけです。

たとえば自作のルーチンがMYINTというラベルのアドレスにあるとすると、

```
E809:    LD    HL, MYINT
```

と書き換えます。ただし、書き換えるのにモニタのSコマンドなどで書き換えると、書き換える途中にもVRTC割り込みはかかっていますから、アドレスがくるって暴走してしまいます。したがって、あらかじめ自作のルーチンを用意したあと、機械語プログラムで次のようにして書き換えます。

```
        DI
        LD    HL, MYINT
        LD    (0E80AH), HL
        EI
        RET
```

このプログラムを実行したとたんにVRTC割り込みは自作のルーチンを通るようになります。

自作のルーチンは次のようにしておきます。

```
MYINT:  DI              ;念のためDIをかける
        PUSH  DE        ;レジスタをPUSHする
        PUSH  BC        ;HLとAFは既にPUSHしてある
            :
          処理
            :
        POP   BC        ;レジスタをPOPする
        POP   DE
        JP    3080H     ;VRTC本来の割り込みルーチンへ
```

割り込みコントロール回路へ割り込みレベルを出力したりするのはVRTC割り込み本来のルーチンが行なってくれます。そのかわり、DI状態で処理をおこなわなければなりません。したがってそのあいだ他の割り込みがかかりませんから、処理にあまり長い時間をかけるわけにはいきません。

第15章　ランダムテクニック

15-1　XFILESでクランチを

システムディスクに入っているディスクユーティリティの「TRANSFER FILES」は異なるタイプのディスク間でのバックアップなどに使われますが、同じタイプのディスクであっても利用法があります。

多数のファイルの生成、削除を繰り返していると、ファイルデータがディスク上のあちこちに分散してしまいます。このようなディスクを「TRANSFER　FILES」で他のディスクに移すと、一ヶ所にかたまって記録されるようになります。こうすると、ファイルへのアクセス時にヘッドがあまり動く必要がなく、この分だけアクセスが早くなります。

15-2　プリンタの行ごとの印字ずれの対処法

プリンタがロジカルシークモードになっている場合には、印字方向によって行ごとにずれが生じる場合があります。グラフィックキャラクタで表を作成したものを印字する時など、印字ずれが起きては困りますね。

これに対して、PC-8023(C)ではインクリメンタルモード、他のプリンタでは片方向印字モードにすることによって、ずれを防ぐことができます。

PC-8023(C)の場合

```
LPRINT  CHR$(27) ; CHR$(91)
```

他のプリンタの場合

```
LPRINT  CHR$(27) ; CHR$(62)
```

これらのモードの解除は、どちらの場合も

```
LPRINT  CHR$(27) ; CHR$(93)
```

とします。なお、プリンタがどれを使うかわからない場合は、上の2つとも実行しておけばよいでしょう。

15-3　LSET，RSET，MID$の利用法２つ

LSET，RSETはFIELD文で宣言された文字変数に対して用いることになっていますが、実は普通の文字変数に対しても使用できます。ただし、あらかじめその文字変数にはダミーの文字列を入れておく必要があります。

```
A$ = "ABCDEF"          (A$を6文字長の文字変数として宣言する)
LSET A$ = "PQR"
```

とすると、A$にはPQRが入り、残りの３文字にはスペースが詰められます。RSETで同じ

ことを行なうと、初めに3文字のスペースが入り、そのあとにPQRが入ります。文字列の長さは変りません。考えてみると、これはMID$文の働きとよく似ています。たとえば、

A$　=　" ABCDEF"

MID$(A$, 1)=" PQR"

とすると、A$はPQRDEFとなります。代入しなかった所がスペースになるかそのまま残るかの違いだけです。

この3つの文に共通する特徴として次のものがあります。ふつう、文字列を操作するとそのたびに古い文字列は捨てられてメモリ上に新しく文字列が作られ、メモリがいっぱいになったところでガーベジコレクションが起こります。しかし、LSET, RSET, MID$を使って操作すると、古い文字列の上に新しい文字列が上書きされますのでメモリがいっぱいになることがなく、ガーベジコレクションが起こりません。ただしこのためにはプログラム中の全ての文字列代入操作をこの方法でおこなう必要があり、INPUT文なども使えないという制限があります。またあらかじめ宣言しておいた以上の長さの文字列が代入ができない(余りは無視される)という欠点もありますが、絶対にガーベジコレクションを起こしたくないときにはやむを得ないでしょう。(なお、通常は問題ないのですが、LSET, RSETでは、代入される文字変数のディスクリプタが指すアドレスが8000H〜E879Hにないと、上書きされずに新しく文字列がつくられてしまいます。MID$ではラベル領域〜配列変数領域を指しているときにかぎって新しく文字列がつくられますが、こんな使い方はしないでしょう。)

さて、これらの文の別の利用法に、ディスクリプタを操作して、指定したメモリにデータを書き込むというのがあります。次の例を見てください。

```
10 WIDTH 80,25 : LOCATE 0,10
20 A$=SPACE$(80)
30 P=VARPTR(A$)
40 POKE P+1,&HC8 : POKE P+2,&HF3
50 '
60 MID$(A$,1)="LINE0"
```

まず20行で80文字長の文字変数を宣言し、30, 40行でディスクリプタを操作してA$が画面の最上行を指すようにします。そして60行でA$にLINE 0という文字列を代入すると、画面の左上にその文字列が現われます。(ディスクリプタの指すアドレスがE879Hよりも後にありますので、LSET, RSETは使えません。)この方法を用いると画面上の複数の特定の場所に文字列を表示するのに、いちいちLOCATE文で指定する必要がなくなります。次の例ではキーボードから文字を入力しながら入力した文字数を画面の右上に表示し続けます。このプログラムをMID$を使わずに書くと、POS(0)とCSRLINを使っためんどくさいものになります。

```
100 WIDTH 80,25 : CONSOLE 1,25 : CLS
110 A$=SPACE$(80)
120 P=VARPTR(A$)
130 POKE P+1,&HC8 : POKE P+2,&HF3
140 COUNT=0
150 '
160 WHILE 1
170   MID$(A$,70)=STR$(COUNT)
```

```
180    PRINT INPUT$(1);
190    COUNT=COUNT+1
200 WEND
```

15-4　配列データ高速読み込み

数値型の配列のデータを読み込むには、通常次の方法があります。

①プログラム中にデータ文として存在するデータをREAD文で読む。

②ファイルとして存在するデータをINPUT＃n文で読む。

どちらの方法をとったとしてもデータが大量な場合には結構時間がかかります。このような場合には、データを機械語ファイルとしてしまっておいて、BLOAD文で一気に読み込むことができます。

8000 ↑
単純変数テーブル
配列変数テーブル
BSAVE
BLOAD
↓ FFFF

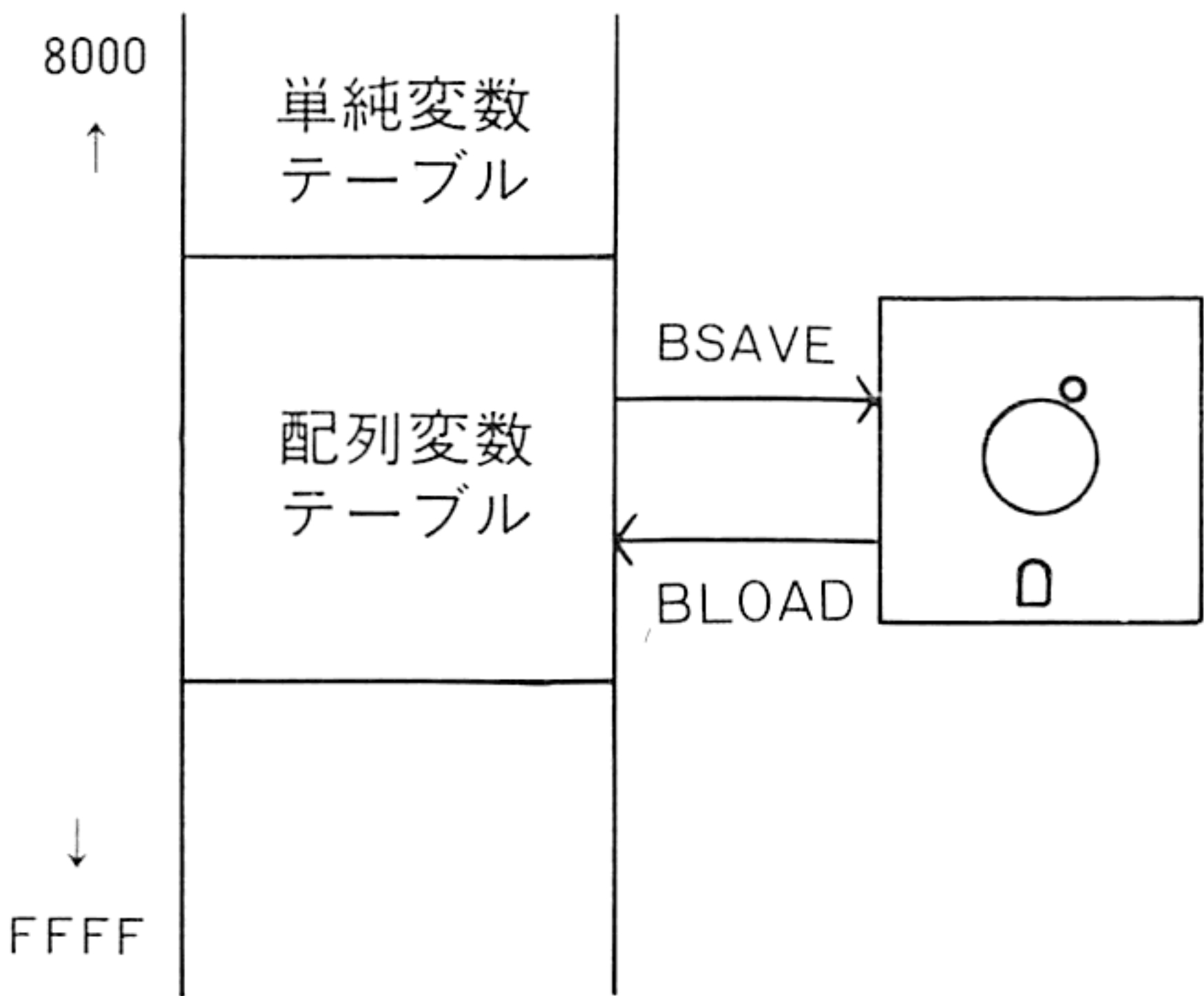



図15－A　配列データ高速読み込み

まず配列データをBASVEします。開始番地はVARPTR(＜配列名＞(0))です。1次元配列の時OPTION BASEの値はEC1FH番地に入っていますから、VARPTR(＜配列名＞(PEEK(&HEC1F)))としても結構です。データの長さは、(要素の数)×(1要素のバイト数)です。

(要素の数)は、

「添字の最大値＋1－OPTION　BASE」をすべてかけ合わせたものです。

(1要素のバイト数)は、

整数型……2バイト

単精度……4バイト

倍精度……8バイト

です。たとえば、DIM　A%(5, 6, 7)、OPTION　BASE 0の時は、

BSAVE"＜ファイルディスクリプタ＞",VARPTR(A%(0,0,0), 6＊7＊8＊2

となります。

データをBLOADする時も同じです。まずDIMでBSAVEした時と同じ大きさの配列を宣言します。次にBLOADします。先ほどの例では

BLOAD"＜ファイルディスクリプタ＞", VARPTR(A%(0,0,0))

となります。

15-5　実行中にプログラム自身を書き換える

プログラム実行中にプログラム自身を書き換えたい場合があります。たとえばデータ文を自動的に作成して、そのデータを使って処理をするときや、入力した式を使って計算をするときなどです。こんなことはできないような気がしますが、実はN88DISK-BASICにはそういう機能があるのです。それはCHAIN文です。書き加えたいプログラムを一旦ディスクにデータファイルとして書き出し、そのファイルをCHAIN MERGEを使って取り込めばよいわけです。ただし、ALLオプションを付けてもDEF FN, OPTION BASE, ON ERROR GOTO, ON 割り込みGOSUBなどは無効になりますから、CHAIN MERGEした後で設定しなおす必要があります。下の例はLINE INPUTした式を計算して結果を表示するものです。

```
10 LINE INPUT "SHIKI: ",SHIKI$
20 OPEN "SHIKI.asc" FOR OUTPUT AS#1
30 PRINT #1,"60 ans=";SHIKI$
40 CLOSE
50 CHAIN MERGE "SHIKI.asc", 60, ALL
60 ANS=5*6
70 PRINT SHIKI$;" = ";ANS
80 END
Ok

run
SHIKI: 1*2*3/4
1*2*3/4 =  1.5
Ok
```

15-6　FIX, INT, CINT

この3つはどれも実数型を整数に変換する関数ですが、微妙な違いがあります。

●FIX…整数部分をとる関数で1.5→1, -2.3→-2となります。

●INT…その値よりも小さい整数の中で最大のものをとる関数です。1.5→1となりますが、-2.3→-3となります。

●CINT…小数部分を四捨五入します。1.5→2, -2.3→-2となります。

これらを図に示すとこのようになります。

-3 -2 -1 0 1 2 3

INT FIX CINT INT CINT FIX FIX INT CINT FIX INT CINT

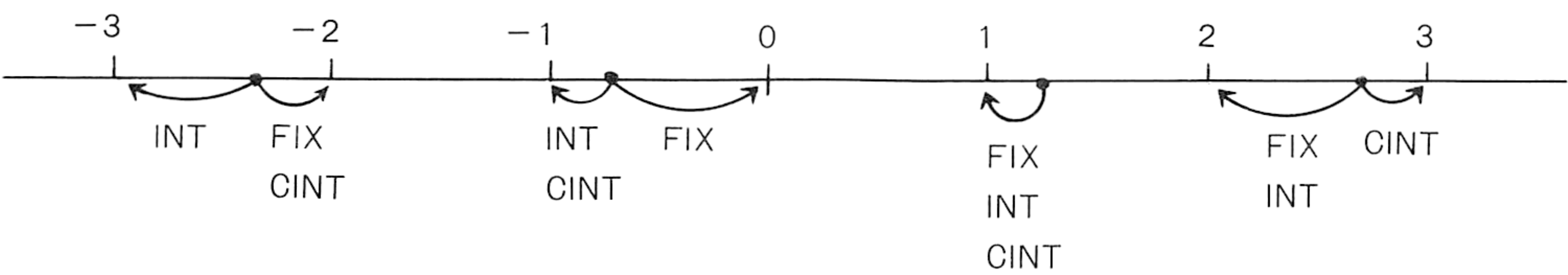



図15－B　整数変換

15-7　数値と文字列との変換

数値を文字列に変換する関数にはSTR$をはじめとして、HEX$, OCT$, CHR$, MKI$, MKS$, MKD$とたくさんありますが、それらの一覧表を上げます。

数値	変換	文字列	備考
数値	→ STR$ / ← VAL	プリントする形式で文字列に	1文字目にはスペースor負号が入る
数値	→ HEX$ / ← VAL("&H"+XXXX$)	16進数に	
数値	→ OCT$ / ← VAL("&O"+XXXX$)	8進数に	
数値	→ CHR$ / ← ASC	数値をキャラクタコードとする文字に	0≦数値≦255
数値	→ MKシリーズ / ← CVシリーズ	内部表現をそのまま文字列に	ランダムファイルのため。文字列の長さが一定である。

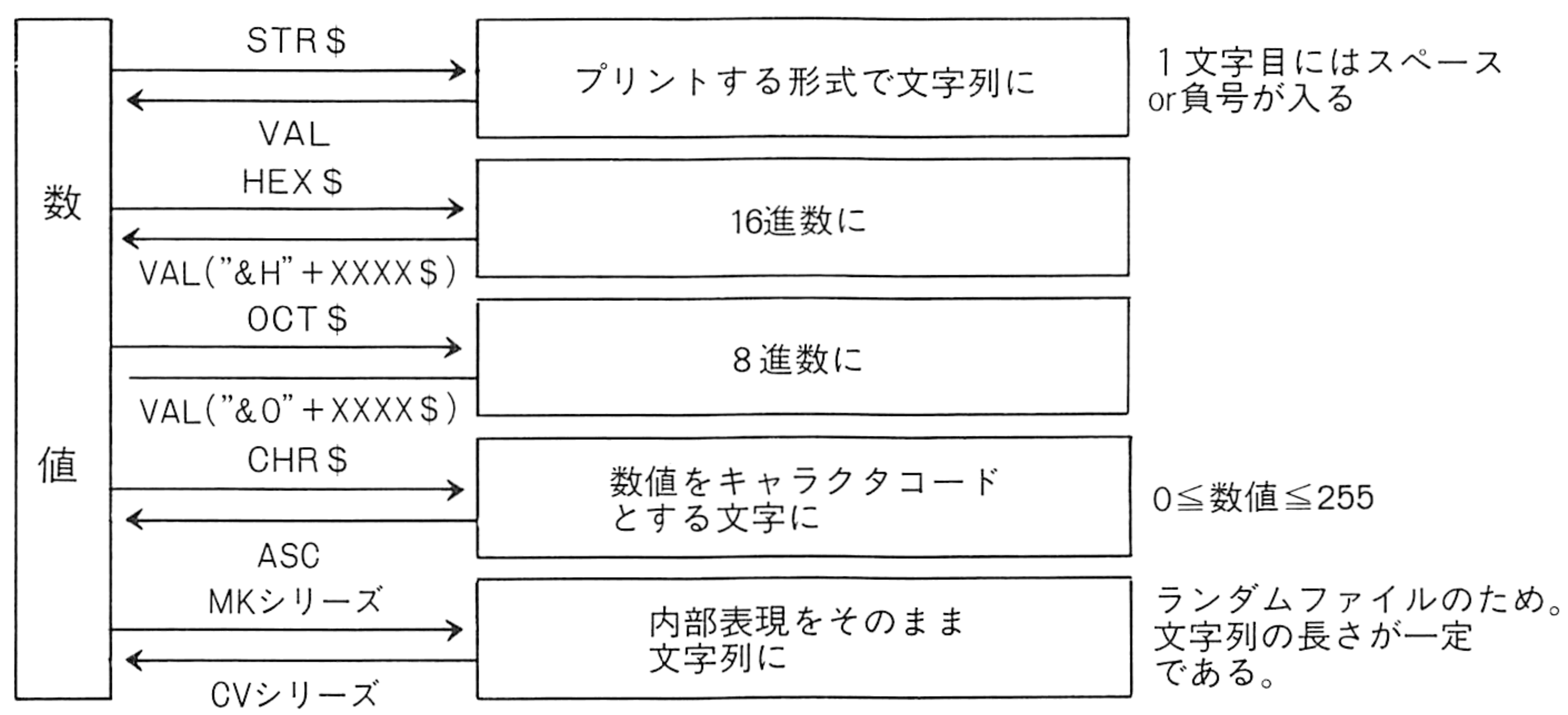



図15－C　文字列への変換

15-8　数値の内部表現

CALL文では変数のアドレスが渡されますが、変数の内部表現がわからなければ利用のしようがありません。またVARPTRを用いて変数の値を操作するときも同様です。そこで数値の内部表現を下に示します。

〔整数型〕

VARPTRが指す

02 | 変数名 | □□

上位　下位

15 14 13 12 11 10 9 8　7 6 5 4 3 2 1 0

符号
0…正の数
1…負の数

数値は符号付きの16ビット2進数で表わされています。
10進数との対応は、16ビットを4ケタの16進数に変換したものに&HをつけてPRINTするとわかります。

16進	10進
7FFF……	32767
7FFE……	32766
⋮	
0001	1
0000	0
FFFF	−1
⋮	
8001	−32767
8000	−32768

〔単精度型〕

VARPTRが指す

04 | 変　数　名 | □□□□

仮数部　指数部

下 | 中 | 上 | ・

23 22 21 20 19 18 17 16　15 14 13 12 11 10 9 8　7 6 5 4 3 2 1 0

この23ビットで小数部を表わす。

m＝1.

1桁上(第23ビット)に1があり、小数点がその間にあるとみなす。

符号(仮数部)
0…正の数
1…負の数

7 6 5 4 3 2 1 0

81Hを0とする指数e

FF	126
FE	125
82	1
81	0
80	−1
7F	−2
⋮	⋮
2	−127
1	−128
0	*

$value = \pm m \times 2^{e}$

数値は仮数部24ビット、指数部8ビットで表わされています。

＊指数部が0であると仮数部に関係なく値は0とみなされます。

〔倍精度型〕

VARPTRが指す

08 | 変数名 | 下 □□□□□ 上 | □

仮数部　指数部

仮数部が7バイト（56ビット）に増えただけで単精度と同じです

図15－D　数値の内部表現

15-9　三角関数の求値法

N88-BASICインタプリタがどのようにして数値関数の値を求めているのかというと、すべて近似式によって計算しているのです。ここではN88-BASIC，N-BASICが使用している三角関数(単精度)を求めるアルゴリズムを紹介します。

① SIN(x)

$w \leftarrow x/2\pi$

$u \leftarrow w - INT(w)$

もし、

$0 \leqq u < 0.25$　のとき　$v \leftarrow u$

$0.25 \leqq u < 0.75$　のとき　$v \leftarrow 0.5 - u$

$0.75 \leqq u < 1$　のとき　$v \leftarrow u - 1$

$s \leftarrow v^2$

$SIN(x) = v \times (a_1 s^4 + a_2 s^3 + a_3 s^2 + a_4 s + a_5)$

ただし、$a_1 = 39.71067$

$a_2 = -76.57498$

$a_3 = 81.60223$

$a_4 = -41.34167$

$a_5 = 2\pi$

これらは、最良近似多項式の係数ですが、ほぼTaylor展開に一致します。

② COS(x)

$COS(x) = SIN(\frac{\pi}{2} - x)$

③ TAN(x)

$TAN(x) = SIN(x)/COS(x)$

④ ATN(x)

もし、

$x < 0$のとき　$u \leftarrow x$,　$ATN(x) = -ATN(u)$

$x \geqq 0$のとき　$u \leftarrow x$,　$ATN(x) = ATN(u)$

もし、

$u < 1$のとき　$v \leftarrow u$,　$ATN(u) = ATN(v)$

$u \geqq 1$のとき　$v \leftarrow \frac{1}{u}$,　$ATN(u) = \frac{\pi}{2} - ATN(v)$

$s \leftarrow v^2$

$ATN(v) = v \times (a_1 s^8 + a_2 s^7 + a_3 s^6 + a_4 s^5 + a_5 s^4 + a_6 s^3 + a_7 s^2 + a_8 s + a_9)$

ただし、$a_1 = 2.866226 \times 10^{-3}$

$a_2 = -1.616574 \times 10^{-2}$

$a_3 = 4.290961 \times 10^{-2}$

$a_4 = -7.528964 \times 10^{-2}$

$a_5 = 0.1065626$

$a_6 = -0.1420890$

$a_7 = 0.1999355$

$a_8 = -0.3333315$

$a_9 = 1$

図15－E　三角関数の求値法

15-10　モニタで他のROMを見る方法

モニタでは普通N88-BASICインタプリタROMしか読めませんが、内部的には他のROMも読めるようになっています。この切り換えスイッチはF1E1H番地で、ここに、0，1，3のどれかを入れてROMを選択します。

0　……　N88-BASIC ROM

1　……　N-BASIC ROM

3　……　4 th ROM＃1

15-11　モニタでテキストRAMを見る方法

上の3種類のROMの内容を読むルーチンはRAM上にあります。ですからそのルーチンを書き換えるとテキストRAMの内容を読むこともできます。ここでは4thROM用のルーチンを書き換えてみます。このプログラムを実行してみてください。

```
100 '
110 '    read text ram in monitor
120 '
130 DEFINT A-Z
140 AD=&HF2C0
150 '
160 FOR I=AD TO AD+&H2E
170    READ D$ : POKE I,VAL("&H"+D$)
180 NEXT
190 '
200 SLAD=AD+&H23 : P=VARPTR(SLAD)
210 POKE AD+2,PEEK(P) : POKE AD+3,PEEK(P+1)
220 P=VARPTR(AD)
230 MONH=&HE66C : POKE MONH,&HC3
240 POKE MONH+1,PEEK(P) : POKE MONH+2,PEEK(P+1)
250 END
260 '
270 DATA F1,21,23,D0,11,A0,F1,01,0C,00,ED,B0,3E,03,32,E1
280 DATA F1,3E,48,32,CE,F1,AF,32,CF,F1,32,F4,F1,32,DE,F1
290 DATA C3,43,60,F3,3A,C2,E6,F6,02,D3,31,7E,C3,6C,F1
```

機械語プログラムはF2C0H番地から格納されますが、別の場所がよいときには140行を書き換えてください。このプログラムでは、モニタにはいるときにモニタの4thROMを読むルーチンを自動的に書き換えるルーチンを付け加えます。また、自動的にF1E1H番地に3を入れる機能もあります。モニタにはいって0番地あたりを読んでみてください。たしかにテキス

トRAMが読まれていますね。なおこの機能を止めるときにはPOKE &HE66C，&HC9としてください。

15-12　1／4角文字のフォントスペアとユーザー定義キャラクタ

旧PC-8801のN88-BASIC Ver1.1以上では、漢字ROMがなくてもJISコード100H-17FHの1/4角文字をコード200H-27FHとして、PUT KANJI文で出すことができました。この機能はPC-8801mkⅡでは漢字ROMが初めからついていますので必要ないのですが、互換性のためか残っています。ちょっとやってみましょう。

```
10 SCREEN 0,0
20 FOR CODE=&H200 TO &H2FF
30   PX=(CODE AND &H1F)*16
40   PY=((CODE AND 255) ¥ 32)*16
50   PUT@(PX,PY), KANJI(CODE), PSET
60 NEXT
```

これを実行すると、画面に1/4角文字が出力されます。でも形が少し違いますね。何かに使えるかもしれません。ところで下半分には変なものがPUTされています。これはコード280H-2FFHに対応する文字です。なぜこんなものが出るのでしょうか。

これらの1/4角文字のフォントはN-BASIC ROMのアドレス7B00H-7EFFHにコード200H-27FHの128文字分(1文字8バイト)用意されています。ですから、コード280H-2FFHを指定すると、この後のアドレス7F00H-82FFHの内容が文字フォントとして使われてしまったわけです。

さて、漢字フォントを読み出すときは一時的にメモリをN-BASICモードに切り換えてから読み出します。ですからアドレス8000H-82FFHはウインドウではなくてRAMです。しかもそのアドレスのメモリはN88-BASICでは使っていません。したがってここに文字のフォントデータを作っておけば、PUT KANJI文で出力できます。ユーザー定義キャラクタと言ってよいかもしれません。計算してみると、アドレス8000Hはコード2A0Hにあたりますから、2A0H-2FFHの96文字に定義できます。

フォントデータをモニタで書き込むときは、ポート70Hに80HをOUTすると、8000HからのRAMそのものに書き込むことができます。フォントとデータとの対応は次のようになっています。

フォント (MSB … LSB)	アドレス (コード2A0Hのとき)
○●●●●○○○	8000H
●○○○○●○○	8001H
●○○○○○○○	8002H
●○○○○○○○	8003H
●○○○○○○○	8004H
●○○○○●○○	8005H
○●●●●○○○	8006H
○○○○○○○○	8007H

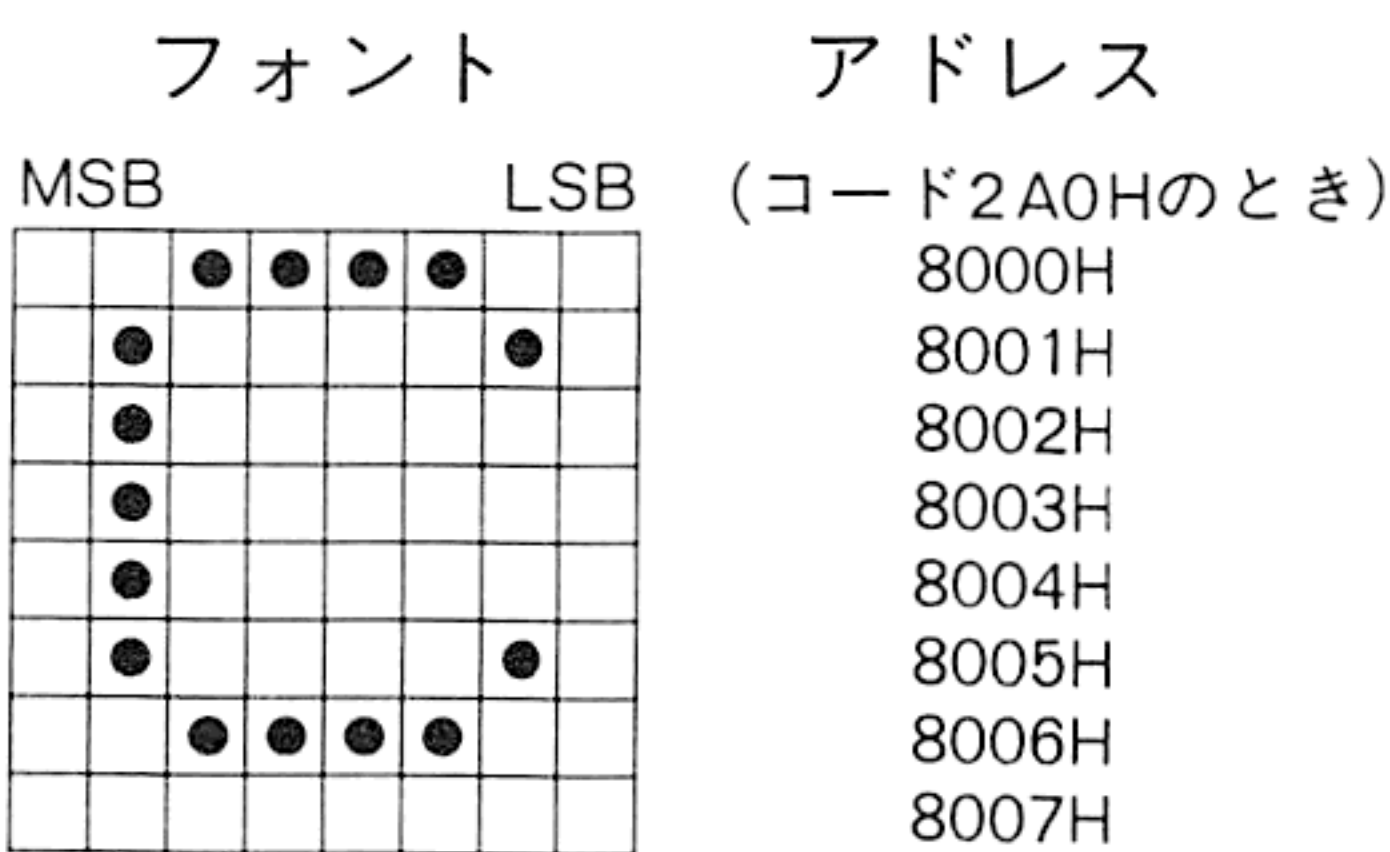



図15-F　フォントとデータとの対応

15-13　N-BASICモードからN88-BASICモードへ

N88-BASICモードからN-BASICモードへはNEW ON 1で切り換えられますが、逆の働きをするコマンドはありません。そこでN-BASICモードからN88-BASICモードへソフトで移る方法を紹介しましょう。

(1)ディップスイッチがN88-BASICモードのとき

これは簡単でたいていOUT　&H31, 0だけですみます。これはポート31Hに0を出力するとROMがN88-BASICに切り換わるからです。ROMを切り換えただけなので当然暴走しますが、暴走するとほとんどの場合リセットがかかるのでN88-BASICが起動するというわけです。確実に切り換えようとするなら(2)の機械語プログラムをつかいます。

(2)ディップスイッチがN-BASICモードのとき

この場合はROMを切り換えるだけではすみません。なぜなら一旦N88-BASICモードになっても、その起動ルーチンでディップスイッチがN-BASICモードになっているのを読み取り、N-BASICモードに切り換えてしまうからです。そこでそのルーチンをうまくごまかさなくてはなりません。

```
F3          DI
AF          XOR   A
D3 31       OUT   (31H), A
DB 30       IN    A, (30H)
F6 03       OR    3
C3 FD 77    JP    77FDH
```

このプログラムを使うとディップスイッチの位置にかかわりなくN88-BASICモードに切り換わります。このプログラムを適当な場所(空いている場所ならどこでもよい)に書き込んで実行させるとN88-BASICに切り換わります。

15-14 N-BASICでフリーエリアを7.5K増やす

N-BASICモードでは、テキストRAMは使われていません。そこで、この余っているRAMをつかってN-BASICモードでのフリーエリアを増やす方法を考えてみましょう。

PC-8001の場合は次のプログラムで簡単に行えましたが、PC-8801mkⅡでは、6000H番地以降にもN-BASICインタプリタの一部が置かれているため、このままでは使えません。

（PC-8001＋PC-8011の場合）

```
C000 21 00 00 11 00 00 01 00
C008 60 ED B0 3E 60 32 DA 17
C010 D3 E2 C3 00 00


*GC000
```

PC-8801のN-BASICインタプリタの拡張部分は、約280バイトです(N-BASICVer1.1の場合)。

そこで、右の図にあるように、この拡張部分を、6000H番地から、61FFH番地の間へリロケートすることにより、6200H番地から6FFFH番地までの7.5Kバイトをフリーエリアとして使えるようにします。次に示すプログラムはこの処理を行なうものです。

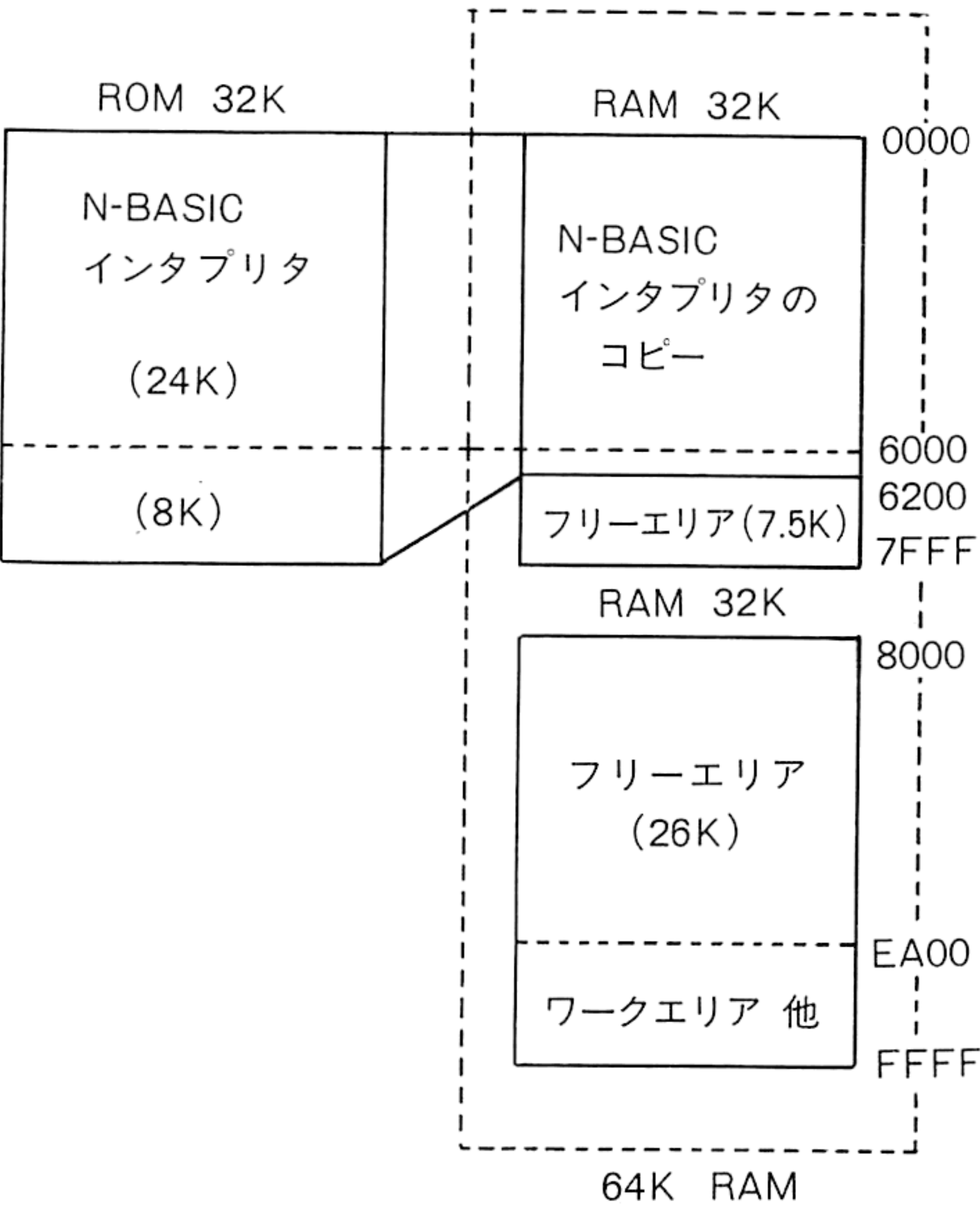



図 N-BASICのフリーエリア

```
100 '
110 '---- PC-8801mkII  ( N-BASIC MODE +7.5K ) ----
120 '
130 RESTORE 190
140 FOR I=&HFF50 TO &HFF62
150   READ DA$ : POKE I,VAL("&H"+DA$)
160 NEXT I
170 DEF USR=&HFF50 : DM=USR(0)
180 '
190 DATA 21,00,00,11,00,00,01,00,60,ED,B0,C9,3E,02,D3,31
200 DATA C3,00,00
210 '
220 RESTORE 270
230 FOR I=&H6000 TO &H611F
240   READ DA$ : POKE I,VAL("&H"+DA$)
250 NEXT I
260 '
270 DATA 3E,0B,CD,7C,01,3E,07,CD,83,01,3E,EF,CD,83,01,AF
280 DATA CD,83,01,3E,01,CD,83,01,CD,E9,01,2F,E6,F0,FE,10
290 DATA C9,3E,0F,18,01,AF,F5,3A,C9,ED,FE,FB,28,0F,CD,00
300 DATA 60,20,0A,3E,17,CD,7C,01,F1,CD,83,01,F5,F1,C9,CD
310 DATA 25,60,11,00,C0,C9,3A,C7,ED,B7,28,11,3E,91,CD,29
320 DATA 02,3E,04,32,CB,ED,AF,CD,7C,01,CD,21,60,3A,55,EB
330 DATA C9,AF,32,22,EF,C3,CF,0A,0E,0A,21,22,EF,79,32,C2
340 DATA ED,CD,C3,60,C2,66,10,78,FE,01,9F,20,03,32,C2,ED
350 DATA 21,76,EA,C3,58,10,FE,50,38,15,21,47,60,4F,06,00
360 DATA 09,7E,A7,37,C8,A7,C9,09,1F,1D,00,00,2D,2F,00,CD
370 DATA A1,10,D8,FE,41,38,0C,FE,5B,38,0A,FE,61,38,04,FE
380 DATA 7B,38,02,A7,C9,F5,3A,23,EF,A7,20,04,F1,EE,20,C9
390 DATA F1,A7,C9,3E,0A,B9,C2,80,10,DB,0A,E6,80,32,23,EF
400 DATA 16,7F,C3,89,10,D5,11,68,0A,CD,95,40,20,09,DB,40
410 DATA E6,02,20,03,3E,22,11,3E,02,D3,31,DB,40,E6,02,20
420 DATA 04,11,94,56,19,D1,3A,67,EA,C3,FC,09,B7,8B,98,6F
430 DATA 58,5F,89,93,73,38,F5,3A,55,EA,D3,E4,F1,FB,C9,CD
440 DATA 02,16,C3,3B,17,00,00,00,00,00,00,00,00,00,00,00
450 '
460 POKE   &H84,&HCD
470 POKE   &H85,&H46 : POKE   &H86,&H60
480 POKE  &H9FA,&HD5 : POKE  &H9FB,&H60
490 POKE  &HB18,&HCD
500 POKE  &HB19,&H3F : POKE  &HB1A,&H60
510 POKE  &HFD9,&H68 : POKE  &HFDA,&H60
520 POKE  &HFEC,&H86 : POKE  &HFED,&H60
530 POKE &H1710,  &HF : POKE &H1711,&H61
540 POKE &H179F,&H61 : POKE &H17A0,&H60
550 POKE &H17F9,  &H6 : POKE &H17FA,&H61
560 POKE &H187E,  &H6 : POKE &H187F,&H61
570 POKE &H1880,  &H6 : POKE &H1881,&H61
580 POKE &H17DA,&H62
590 POKE &H1850,&H33
600 '
610 DEF USR=&HFF5C : DM=USR(0)
620 '
630 END
```

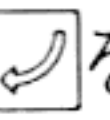
N-BASICでのこのプログラムを実行すると、リセットがかかって、次の画面が表示されます。

```
NEC PC-8001 BASIC Ver 1.3
Copytight 1979 (C) by Microsoft

Ok
```

フリーエリアの大きさを見てみましょう。

```
print fre(0)
 34466
Ok
```

確かに、7.5Kバイト増えていまね。

この例はディスクがない場合ですが、DISK-BASICであってもこのまま使うことができます。ドライブ1にシステムディスクをセットしてこのプログラムを実行して下さい。

```
Two surface disk version (20-Sep-1981)
How many files(0-15)? 3
NEC PC-8001 BASIC Ver 1.3
Copyright 1979 (C) by Microsoft

Ok
print fre(0)
 27029
Ok
```

これで大きなプログラムもDISK-BASICで走らせることができるようになります。

《注意》

- このモードでは、N-BASICインタプリタがRAM上にあります。もし、この部分が破壊されたりすると暴走する危険がありますので注意して下さい。
- リセットボタンを押すと(ホットスタートも含む)もとのROMバージョンのN-BASICに戻ってしまいます。このモードのままでリセットしたい場合(新たにDOSを読み込むときなど)は、モニタモードでG 0⏎を実行して下さい。

15-15 ドライブ2を片面として使う

旧PC-8801では片面ドライブ(PC-8031)を接続することができました。PC-8801mkIIでは接続することはできませんが内蔵ドライブを片面ドライブとして扱うことができます。これはサブシステムには片面モードというのがあり、PC-8031と同じ動作をするようにできるからです。そこで、N88-BASICのワークエリアを操作して、PC-8031が接続されているようにみせかければ片面ディスクも扱えるようになるわけです。

まずドライブ2を片面モードにします。次のプログラムを実行してください。サブシステムとハンドシェークを行なって(第10章サブシステム参照)コマンドを送っています。

```
100 '
110 '    set single serface mode
120 '
130 '
140 PA=&HFC: PB=&HFD: PC=&HFE: CW=&HFF
150 '
160 A=&H17 : GOSUB *OUT.CMD
170 A=&HD  : GOSUB *OUT.DAT
180 '
```

```
190 END
200 '
210 *OUT.CMD
220  OUT CW,15
230 *OUT.DAT
240  IF (INP(PC) AND 2)=0 THEN 240
250  OUT CW,14
260  OUT PB,A:OUT CW,9
270  IF (INP(PC) AND 4)=0 THEN 270
280  OUT CW,8
290  IF INP(PC) AND 4 THEN 290
300 RETURN
```

次に、ワークエリアのドライブタイプ対応表を操作して、片面ディスクが接続されているようにみせかけます。

```
poke &Hef64+peek(&Hef60)+peek(&Hef61)+1,2
```

もちろんこの2つは一つのプログラムにしてかまいません。これでドライブ2が片面ドライブになりました。ドライブ1は両面のままです。ドライブ2に片面ディスク(PC-6001mkⅡのディスクなど)を入れて使用できます。またこの方法はPC-8801＋PC-80S31でも使えます。頻繁に両面/片面の切り換え行なう場合は、「10-4-2(P 215)」に、同じことを機械語で行ない、未使用コマンド(ISET)を使って自由に切り換えられるようにしたユーティリティがありますので、そちらを参照して下さい。

15-16 音階をだす方法

PC-8801mkⅡでは拡張命令のCMD SINGでメロディーを出すことができます。これは内蔵のスピーカーを目的の周波数で振動させることによって実現しています。このことについて説明するまえに、PC-8801mkⅡでスピーカーを制御する2つの方法を説明しておきます。

まず最初は旧PC-8801と同じ方法で、BEEP文が使っています。PC-8801mkⅡは2400Hzの音源を持っていて、これをOUT命令でスピーカーにつなぐと2400Hzの音がでます。この制御にはI／Oポート40Hのビット5を用います。

ポート 40H (OUT) N88-BASICのワークエリア：E6C1H番地

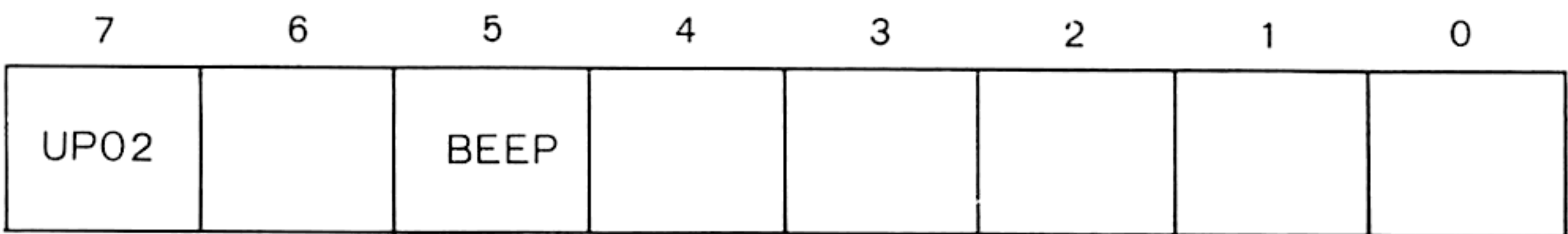
7	6	5	4	3	2	1	0
UPO2		BEEP					

他のビットも使っています

図15 - G　BEEPの制御

このビットを1にすると2400Hzの音がでます。ちょっとやってみましょう。ポート40Hの他のビットを変更してはいけませんから、ワークエリア(E6C1H番地)を参照して他のビットの値を得ます。

```
10 A=PEEK(&HE6C1) OR &H20      :'ビット5を1にする
20 POKE &HE6C1,A               :'ワークエリアをセットする
30 OUT &H40,A                  :'ポートにOUTして音を出す
```

これはBEEP 1と同じですから、音を止めるときにはBEEP 0とします。

次にCMD SINGで使用している方法を説明します。これは同じポート40Hのビット7で制御します。今度は特別な音源は持たず、周波数はソフトで決定します。つまり、このビットがスピーカーに直接つながっていて、このビットをたとえば600HzでON／OFFすると、600Hzの音がでるというわけです。

```
10 A=PEEK(&HE6C1)              :'現在のポート40Hの状態を得る
20 A=A OR &H80                 :'ビット7を1にする
30 POKE &HE6C1,A : OUT &H40,A
40 A=A AND &H7F                :'ビット7を0にする
50 POKE &HE6C1,A : OUT &H40,A
60 GOTO 20
```

これはBASICで書いた例ですのであまり高い音はでません。また、機械語で行なう場合も、DMAや割り込みを止めないと、正確に一定の周波数の音が出せませんから、濁った音になります。なお、機械語で、割り込みをとめた状態で音をだすときには、いちいちE6C1H番地にOUTする値を書き込む必要はありません。

15-17 旧PC-8801でCMD SINGを

旧PC-8801でCMD SINGを使うためには、市販されているmkⅡサウンド基板を購入すれば拡張命令パッケージ以外のソフトにも対応できる(ただし8801と8801mkⅡで音を変えているソフトではROMを読んで区別していますから、8801用の音しかでません)のですが、CMD SINGで音を出すだけであれば、

POKE&HDEE1, &H20

とすればかなり濁った音ですが一応音を出すことができます。これはCMD SINGで使うサウンドポート(ポート30Hビット7)をビット5(BEEPのポート)に変更したのです。

15-18 未使用命令を使う (ユーザー拡張命令)

BASICから機械語ルーチンを呼ぶ場合、USR関数やCALL文を用いますが、USR関数は引き数が1個しか使えない、CALL文は引き数として変数しか使えない、値を返す機能がない(変数を介し値を返すことはできますが)ため式の中で使えない、などの欠点があり、BASICプログラム中で記述する時に必ずしも使いやすくありません。そこで他のBASICの命令と同じ簡便さで機械語ルーチンを呼ぶためのユーザー拡張命令の作り方を紹介します。

これはUSR関数・CALL文に比べて作り方が難しくなっています。BASICで簡単に使えるようになった分のしわ寄せが機械語にきたわけですが、一旦ルーチンを作ってしまえばあとはブラックボックスとして使えばよいわけですから、よりいっそうBASICプログラムに専念できます。

中間言語の所で述べたとおり、N88-DISK-BASICでは使用されていないキーワード(SRQ,

WBYTE, RBYTE, POLL, ISET, IEEE, IRESET, STATUSの８コ)があります。これらはもともとIEEEインターフェース制御用として用意された命令なのですが、通常のN88-DISK-BASICではサポートされていないため空きになっているわけで、SRQを除く7つは、ユーザー独自の命令用のキーワードとして使えます。つまりユーザーが作成した機械語ルーチンを他のBASIC命令とまったく同じに使用することができるようになるわけです。ここではその方法を説明します。本書のあちこちに出て来る「POLL文」もこの方法によって作られていますし、N88-BASICにある「拡張命令」(CMDシリーズ)もCMDというキーワードを使って同じ方法によって実現されています。

15-18-1　ステートメントの作り方

ステートメントとして使用できるキーワードは、WBYTE, RBYTE, POLL, ISET, IRESET, STATUSの６つです。

ステートメント用機械語ルーチンは次のような流れで作ります。

①引き数を評価してレジスタやメモリにしまう。

②テキストポインタ(HLレジスタ)を保存する。

③目的とする処理を行なう。

④テキストポインタ(HLレジスタ)を戻し、RETする。

このうち①の引き数の評価をするというのはけっこう面倒です。これを簡単に実現するためにいくつかのROM内ルーチンやワークエリアを使用します。これらは非常に便利な機能で、独自のステートメントを作る際にはほとんど必須といえるものですから、是非使い方をマスターして下さい。

15-18-2　引き数の評価に使用するルーチン、ワークエリア、レジスタ

(1)浮動小数点アキュムレータ(FAC)

FACはBASICインタプリタのアキュムレータで、すべての計算はここを中心にして行なわれます。FACは次のような構成になっています。

	整数型・文字型	単精度実数型	倍精度実数型
EC3DH			仮数部最下位 バイト
EC3EH			↓
EC3FH			↓
EC40H			↓
EC41H	下位 バイト	仮数部下位 バイト	↓
EC42H	上位 バイト	仮数部中位 バイト	↓
EC43H	(文字型の場合はディスクリプタのある番地を指す)	仮数部上位 バイト	仮数部最上位 バイト
EC44H		指数部 1バイト	指数部 1バイト
EABDH番地の値	2：整数型 3：文字型	4	8

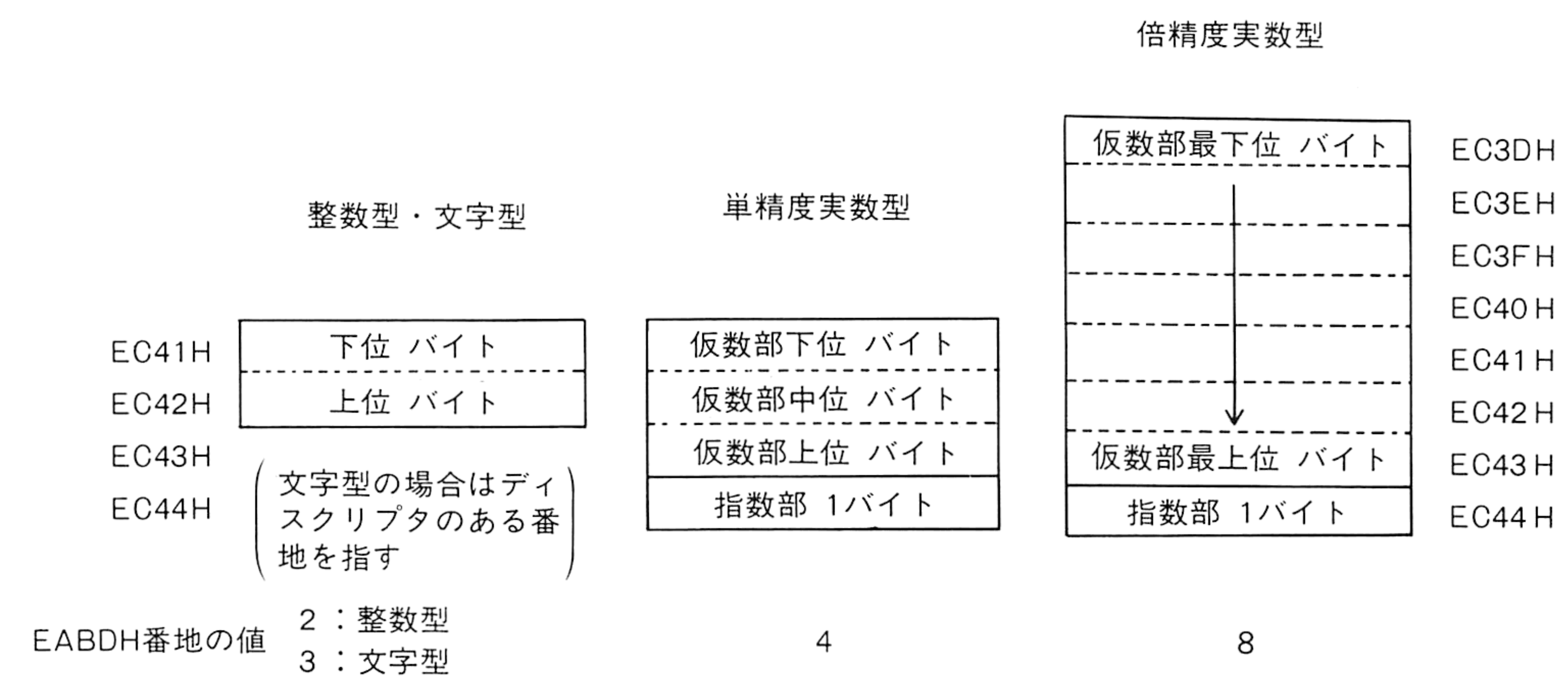



図15－H　FAC

各バイトの意味は「15-8　数値の内部表現」を見て下さい。

(2)HLレジスタ……テキストポインタ

OAR(ポート70H)とともにBASICプログラム(テキスト)中を指し、現在どこを実行しているかを示します。OARがテキスト中の1Kバイトのブロックをウィンドウ(8000H～83FFH)に移し、HLレジスタがウィンドウ内のアドレス(8000H～83FFH)を持っています。ダイレクトモードの時にはワークエリア内の中間言語バッファ(E87AH～E9B7H)をHLレジスタが直接指し、OARは無関係になります。このテキストポインタは単語ゲットや評価などのルーチンの入力条件となります。

(3)44D5H……ウィンドウ内アドレス→実アドレスへの変換

HLに入っているウィンドウ内アドレスとOAR(ポート70H)の値からHLの示す実アドレス(0000～7FFFH)を計算してHLに入れます。HLレジスタ以外のレジスタ、フラグは変化しません。

<例>

HLレジスタ＝8043H

OAR　　　＝10H

のとき、CALL　44D5Hを実行すると

(8043H－8000H)＋1000H＝1043Hですから

HLレジスタ＝1043Hとなります。

(4)44A4H…実アドレス→ウィンドウ内アドレスへの変換

(3)と逆のことを行ないます。同時にOARを設定しなおします。HLレジスタ以外のレジスタ、フラグは変化しません。

<例>

HLレジスタ＝32A4HとしてCALL 44A4Hを実行すると

HLレジスタ＝80A8H (8000H＋A8H)

OAR　　　＝32Hとなります。

(5)RST　10H……単語ゲット

これは最も重要なルーチンです。テキストから1文字または数を読み込みます。HLが指す所の次の文字(または数)を読みます。

＜例＞

A$＝CHR$(68)というテキストはメモリ中では、

41　24　F1　FF　96　28　0F　44　29となっています。

↑　↑　　　　　　↑

A　B　　　　　　C

a)HLレジスタがAを指しているときRST　10Hを実行すると、

Aレジスタ＝24H(「$」)

HLレジスタ＝Bのアドレス

となります。

b)HLレジスタがCを指しているときRST　10Hを実行すると、

Aレジスタ＝0FH(1バイト整数を表わすID)

HLレジスタ＝E69CH

EAC4，5＝0044H(逆順に入っている)

となります。E69CH，EAC4Hについては後で説明します。

RST　10Hの働きをまとめると下図のようになります。

N 16進	Acc	Carry	Zero	HL	EAC2	EAC3	EAC0,1	EAC4～B	備考
0	N	NC	Z	INC	—	—	—	—	行末
1～8	N	NC	·NZ	INC	—	—	—	—	
9，A	INC HL and RST 10H								
B～E	N	NC	NZ	E69C	N	2	HL＋4	a，b，—	アドレス・行番号
F	N	NC	NZ	E69C	N	2	HL＋3	a，0，—	整数(10～255)
10	HL← (EAC0,1)，DEC HL，RST 10H								
11～1B	N	NC	NZ	E69C	N	2	HL＋2	N-11H，0 —	1整数(0～10)
1C	N	NC	NZ	E69C	N	2	HL＋4	a，b，—	整数(256～)
1D	N	NC	NZ	E69C	N	4	HL＋6	a，b，c，d，—	単精度
1E	(EAC2)	affected by (EAC2)		INC	—	—	—	—	
1F	N	NC	NZ	E69C	N	8	HL＋10	a，b，c，d，e，f，g，h	倍精度
20	INC HL and RST 10H								スペース
21～2F	N	NC	NZ	INC	—	—	—	—	
30～39	N	C	NZ	INC	—	—	—	—	数字
3A	N	NC	Z	INC	—	—	—	—	文末
3B～	N	NC	NZ	INC	—	—	—	—	

↓N a b c d e f g h
(HL)

E69C : 1E
D : 10

図15－1　RST　10H

HLはNの前を指していたとして、RST 10Hを行なうと結果がどうなるか示します。表中にあるEAC4H番地～EACBH番地はRST 10HのFACともいえるもので、数を読んだ時にその数の値を格納します。その型は、EAC3H番地に入っています。

Carryというのはキャリーフラグのことで、数字(アスキーコード30H～39H)を読んだ時

にのみ立ちます。

ゼロフラグは00と3A(「：」)のとき、つまり、文末を読んだ時に立ちます。(ただし、”　”の中、DATAの中などの：のときにも立ちます。)

今度はNの欄を見て下さい。Nが9，A，20のときはINC　HLをしてRST 10Hをやりなおします。つまりこの3種の文字はスキップされるわけです。

Nが、10，1Eというのは特別なものです。通常のテキストの中には現われません。これらは数を読んだ後で使用されます。数を読むとHLの値はE69CHになり、テキストポインタの値はEAC0，1H番地にしまわれます。ここでRST 10Hを行なうと、E69CH番地の次には10Hが入っていますから、これに従ってEAC0，1H番地からテキストポインタを得、RST 10Hをやり直します。

なぜこのような面倒くさいことが行なわれているのかというと、「読み直し」のためです。数以外のものを読んだ時にはテキストポインタは1つしか進みませんから、DEC HLを行なってRST 10Hと行なうと同じものが読み直せます。ところが数の場合には2つ以上進みますから、いくつもどせばよいかわかりません。そこで数を読んだ場合でもDEC HLをしてRST 10Hを行なうと読み直しができるように用意されたものが1E，10なのです。

数を読んだ後でDEC HLを行なうとHLの値はE69BHになります。そこでRST 10Hを行なうと、E69CHの内容は1EですからAcc，フラグがセットされ、HLはE69CHとなります。RST 10H用のFACはそのままです。つまり初めて数を読んだ時と同じ状態になっているわけです。

(6)11D3H……式の評価

式の値を計算してFACに入れます。このルーチンを使って引き数を計算するわけです。次のようにして使います。

①式の前をHLで指す

②RST　10Hを行なう

③CALL　11D3Hを行なう

こうすると式の値がFACに入り、HLは式の次の文字を指して、もどってきます。ただし、式にエラーがあるとエラーを出力してBASICコマンド待ち(またはON ERROR GOTO の場所)に行ってしまいます。すべてのレジスタが変化します。

(7)1796H……FACの型変換

FACをAccで示される型に変換します。できない場合にはエラーが出力されます。型に対応する値は(1)FACを見て下さい。すべてのレジスタが変化すると考えた方がよいでしょう。

(8)03B3H……エラー出力

Eレジスタにエラーコードを入れて3B3Hにジャンプすると対応するエラーメッセージを出力してBASICコマンド待ちにもどります。(ON ERROR GOTOがかかっている時にはその場所から実行がはじまります。)引数がまちがっている時などに使います。ジャンプする時

のスタックレベルは気にする必要はありません。

(9)56CCH……ストリングスタックのPOP

文字型の引き数を評価した場合、その文字列のディスクリプタはストリング用のスタックに入れられます。これはそのステートメントの処理が終わるまでにPOPしておかないといけません。そうしないと、その文を10回ぐらい実行した所でString formula too complexエラーが出ます。つまり、スタックがオーバーフローしたということです。このルーチンを呼ぶとスタックをPOPしてくれるだけでなく、そのストリングのエリアをフリーエリアに戻し、ディスクリプタのアドレスをHLに入れてくれます。ただし、これを行なった後別の文字列を処理すると、この文字列は消滅します。

15-18-3　ステートメントのイニシャライズ

未使用のキーワードをステートメントとして使用するためにはそのステートメントが実行された時に、用意された機械語ルーチンへ飛ぶように変更しなければなりません。そうしないとFeature not availableエラーが出ます。

ステートメントが実行されると、各ステートメントにより決まったアドレスへジャンプします。未使用ステートメントの場合は次の場所です。

WBYTE ····EEA1H
RBYTE·····EEA4H
POLL ········EEA7H
ISET··········EEAAH
IRESET····· EEADH
STATUS···· EEB0 H

これらのアドレスには通常JP 4DC1Hという命令が入っていて、これらのステートメントが実行されると4DC 1番地へ行きます。4DC1HへジャンプするとFeature not availableエラーが出るようになっています。見ればわかるとおり、これらのアドレスはRAM上にあります。したがってここに入っているJP 4DC1Hを書き換えて用意したルーチンへ飛ばせばよいわけです。このように拡張のために処理の一部を飛び出させる(ここではワークエリアの中においている)のをフック(HOOK)といいます。

では、ちょっとやってみましょう。用意されたルーチンの絶好の例としてROM内のルーチンを使います。すでにあるステートメントのルーチンを使うわけです。ここではPRINT文を使ってみましょう。PRINT文のアドレスは0E54Hです。POLL文のフックアドレスはEEA7Hですから、ここにJP 0E54Hを入れます。

```
mon

h〕seea7
EEA7 C3- C1-54 4D-0e
h〕^b
Ok
```

これでPOLL文がPRINT文と同じ働きをするようになりました。POLL 1＋2とかPOLL 3＊SIN(0.5)とかやってみて下さい。POLL USINGだって使えますよ。

15-18-4 ステートメント作成例

以上述べてきたルーチンを用いてステートメントを作るわけですが、これらは基本的なルーチンで、他にも便利なルーチンがまだまだあります。それらについては「17-1-1 N88-BASIC インタプリタ」を見て下さい。特にゴシックの部分に重要なものがあります。

さて、ここでは簡単な例として画面POKE命令を作ってみましょう。テキスト画面の(0,0)からの相対アドレスにデータをPOKEする命令です。まず書式を決めます。

書式 POLL＜相対アドレス＞、＜データ＞

＜相対アドレス＞、＜データ＞とも整数型とします。今回は範囲のチェックはしません。
＜データ＞は整数型の下位バイトを用いることにします。

まず引数の評価はこうなります。

```
SPOKE:  CALL   11D3H          ;最初はすでにRST 10Hは行なわれています。
        PUSH   HL
        LD     A, 2
        CALL   1796H          ;FACを整数型にします。
        POP    HL
        LD     DE, (0EC41H)   ;DE←FACします。
        PUSH   DE
        LD     A, (HL)        ;<相対アドレス>の次の文字をとり出します。
        CP     ','            ;コンマかどうか確かめます。
        JR     NZ, ERROR      ;もし違ったらエラ-です。
        RST    10H
        CALL   11D3H          ;<データ>を計算します。
        PUSH   HL             ;テキストポインタをセーブします。
        LD     A, 2
        CALL   1796H          ;FACを整数型にします。
        POP    HL             ;いちおうテキストポインタをもどします。
        LD     A, (0EC41H)    ;FACの下位バイトをとり出します。
        POP    DE             ;<相対アドレス>をもどします。
```

＜相対アドレス＞はDEに、＜データ＞はAccに入っています。残りの部分は簡単です。

```
        PUSH   HL             ;テキストポインタをセーブします。
        LD     HL, (0E6C4H)   ;VRAMのアドレスです。
        ADD    HL, DE         ;HLに絶対アドレスを得ます。
        LD     (HL), A        ;VRAMに書き込みます。
```

```
            POP     HL              ;テキストポインタをもどします。
            RET                     ;処理を終わります。

ERROR:                              ;エラールーチンです。
            LD      E, 2            ;Syntax errorにします。
            JP      03B3H           ;エラー出力します。
```

一応、これらをまとめれば完成です。アセンブルリストを出すと次のようになります。このルーチンはリロケータブルでどこからおいてもかまいません。

```
11D3                EVAL:   EQU   11D3H
1796                CNVTYP:EQU    1796H
03B3                ERROUT:EQU    03B3H

E6C4                VRAMTP:EQU    0E6C4H
EC41                FAC:    EQU   0EC41H


0000                SPOKE:
0000 CDD311                 CALL EVAL
0003 E5                     PUSH HL
0004 3E02                   LD   A,2
0006 CD9617                 CALL CNVTYP
0009 E1                     POP  HL
000A ED5B41EC               LD   DE,(FAC)
000E D5                     PUSH DE
000F 7E                     LD   A,(HL)
0010 FE2C                   CP   ','
0012 2017                   JR   NZ,ERROR
0014 D7                     RST  10H
0015 CDD311                 CALL EVAL
0018 E5                     PUSH HL
0019 3E02                   LD   A,2
001B CD9617                 CALL CNVTYP
001E E1                     POP  HL
001F 3A41EC                 LD   A,(FAC)
0022 D1                     POP  DE

0023 E5                     PUSH HL
0024 2AC4E6                 LD   HL,(VRAMTP)
0027 19                     ADD  HL,DE
0028 77                     LD   (HL),A
0029 E1                     POP  HL
002A C9                     RET

002B 1E02           ERROR:  LD   E,2
002D C3B303                 JP   ERROUT

0030                        END
```

実はこのルーチンには多くのムダがあります。ROMルーチンを有効に使えばもっと短くなります。「第17章　ソフトウェア解析」や「付録1　マシン語ルーチンソースリスト」を見て研究して下さい。自分でROMルーチンのディスアセンブルリストを読んでみるのもたいへん参考になります。特に短かそうなステートメント(POKE, BEEP, MOTORなど)

から読むのがよいでしょう。

さてルーチンは用意できましたからこれをステートメントにしましょう。まず、ルーチンを入力します。一応F220H番地から置くことにします。入力しやすいようにダンプリストをあげます。

```
F220 CD D3 11 E5 3E 02 CD 96 17 E1 ED 5B 41 EC D5 7E
F230 FE 2C 20 17 D7 CD D3 11 E5 3E 02 CD 96 17 E1 3A
F240 41 EC D1 E5 2A C4 E6 19 77 E1 C9 1E 02 C3 B3 03
```

次にフックを書き換えます。

```
mon

h〕seea7
EEA7 C3- C1-20 4D-f2
h〕^b
Ok
```

これでPOLLステートメントができました。

POLL　0，&H40

とすると画面の左上すみに「A」が現れます。

このステートメントを改良してPOLL＜X座標＞，＜Y座標＞，＜データ＞　という書式にして、パラメータの範囲チェックをつけると実際に利用できるステートメントができあがるでしょう。

15-18-5　関数の作り方

関数として使える中間言語は、IEEE，STATUSの２つです。これらの関数を作るときもステートメントを作る時とほぼ同じです。ただし、関数ですから、値を返さなければなりません。値はFACにセットします。したがって関数用機械語ルーチンは次のような流れになります。

①引き数を評価してレジスタやメモリにしまう。

②テキストポインタを保存する。

③その関数の処理を行なう。

④結果をFACにセットする。

⑤テキストポインタをもどし、RETする。

この２つの関数のフックアドレスはこうなっています。

IEEE………EEB9H

STATUS…EEB3H

15-18-6　関数作成の例

ではさっそく関数を作ってみましょう。テキスト画面上の位置を指定すると、その場所のアトリビュートコードを返す関数を考えてます(アトリビュートコードについてはP.81をごらん下さい)。画面は80×25のモードのみを考えます。

書式　STATUS　(<X座標>, <Y座標>)

まず、引き数の評価をします。最初は、すでにRST　10Hが行なわれています。

```
GETATR:  LD     A,(HL)       ;最初の文字を読みます。
         CP     '('
         JR     NZ,ERROR     ;'('でなければエラーです。
         RST    10H
         CALL   11D3H        ;<X座標>を計算します。
         PUSH   HL
         LD     A,2
         CALL   1796H        ;FACを整数型に変換します。
         LD     HL,(0EC41H)
         EX     (SP),HL       ;FACをPUSHし、テキストポインタを取り
                                出します。
         LD     A,(HL)       ;次の文字が','であることを確かめます。
         CP     ','
         JR     NZ,ERROR
         RST    10H
         CALL   11D3H        ;<Y座標>を計算します。
         PUSH   HL
         LD     A,2
         CALL   1796H        ;FACを整数型に変換します。
         LD     HL,(0EC41H)
         EX     (SP),HL       ;FACをPUSHし、テキストポインタを取り
                                出します。
         LD     A,(HL)
         CP     ')'
         JR     NZ,ERROR     ;最後の文字が')'でないとエラーです。
         RST    10H          ;次の文字までRST　10Hしておきます。
         POP    DE           ;<Y座標>をPOPします。
         POP    BC           ;<X座標>をPOPします。
         LD     D,C          ;Dレジスタに<X座標>を移します。
```

これで、DレジスタにX座標、EレジスタにY座標が入りました。さて、指定した場所のアトリビュートコードをとってくる方法ですが、これには便利なROMルーチンがあります。4472H番地です。H，LレジスタにそれぞれX座標＋1、Y座標＋1を入れてここをCALLすると、Bレジスタにその場所のキャラクタコード、Cレジスタにアトリビュートコードを入れてもどってきてくれます。これを使って残りの部分を書くとこうなります。

```
        PUSH    HL              ;テキストポインタを保存します。
        EX      DE, HL
        INC     H               ;HレジスタにX座標+1をいれます。
        INC     L               ;LレジスタにY座標+1をいれます。
        CALL    4472H           ;アトリビュートコードを取り出します。
        LD      L, C            ;アトリビュートコードをHLレジスタに入れ
                                  ます。
        LD      H, 0
        LD      (0EC41H), HL    ;アトリビュートコードをFACに入れます。
        LD      A, 2
        LD      (0EABDH), A     ;FACが整数型であることを示します。
        POP     HL              ;テキストポインタをもどします。
        RET                     ;処理を終わります。

ERROR:  LD      E, 2            ;Syntax errorのコードです。
        JP      03B3H           ;エラーを出力します。
```

これをまとめてアセンブルするとこうなります。

```
03B3            ERROUT:EQU  03B3H
11D3            EVAL:  EQU  11D3H
1796            CNVTYP:EQU  1796H
4472            SCRGET:EQU  4472H

EABD            FACTYP:EQU  0EABDH
EC41            FAC:   EQU  0EC41H

0000            GETATR:
0000 7E                LD   A,(HL)
0001 FE28              CP   '('
0003 203E              JR   NZ,ERROR
0005 D7                RST  10H
0006 CDD311            CALL EVAL
0009 E5                PUSH HL
000A 3E02              LD   A,2
000C CD9617            CALL CNVTYP
000F 2A41EC            LD   HL,(FAC)
0012 E3                EX   (SP),HL
0013 7E                LD   A,(HL)
0014 FE2C              CP   ','
0016 202B              JR   NZ,ERROR
0018 D7                RST  10H
0019 CDD311            CALL EVAL
001C E5                PUSH HL
001D 3E02              LD   A,2
001F CD9617            CALL CNVTYP
0022 2A41EC            LD   HL,(FAC)
0025 E3                EX   (SP),HL
0026 7E                LD   A,(HL)
0027 FE29              CP   ')'
0029 2018              JR   NZ,ERROR
002B D7                RST  10H
```

```
002C D1                 POP  DE
002D C1                 POP  BC
002E 51                 LD   D,C

002F E5                 PUSH HL
0030 EB                 EX   DE,HL
0031 24                 INC  H
0032 2C                 INC  L
0033 CD7244             CALL SCRGET
0036 69                 LD   L,C
0037 2600               LD   H,0
0039 2241EC             LD   (FAC),HL
003C 3E02               LD   A,2
003E 32BDEA             LD   (FACTYP),A
0041 E1                 POP  HL
0042 C9                 RET

0043 1E02        ERROR: LD   E,2
0045 C3B303             JP   ERROUT

0048                    END
```

まず機械語ルーチンを入力します。このルーチンもリロケータブルですので、空いている場所ならどこにでもおけますが、ここではF260H番地から入れることにします。ダンプリストは次のようになります。

```
F260 7E FE 28 20 3E D7 CD D3 11 E5 3E 02 CD 96 17 2A
F270 41 EC E3 7E FE 2C 20 2B D7 CD D3 11 E5 3E 02 CD
F280 96 17 2A 41 EC E3 7E FE 29 20 18 D7 D1 C1 51 E5
F290 EB 24 2C CD 72 44 69 26 00 22 41 EC 3E 02 32 BD
F2A0 EA E1 C9 1E 02 C3 B3 03
```

次にフックを書き換えます。STATUSのフックアドレスはEEB3H番地ですからここにJP F260を入れます。

```
mon

h)seeb3
EEB3 C3- C1-60 4D-f2
h)^b
Ok
```

これでSTATUS関数が使えるようになりました。

これを使ったサンプルプログラムを上げます。このプログラムはまず画面に192個の@を書き、次に画面の端から順にアトリビュートコードをしらべていって、そのカラーコードをその位置にプリントします。

```
100 DEFINT A-Z
110 WIDTH 40,25 : CONSOLE 0,25,,1
120 FOR I=0 TO 191
130   LOCATE RND*37,RND*20
140   COLOR INT(RND*8)
150   PRINT "@";
160 NEXT
170 '
180 FOR Y=0 TO 21
```

```
190    FOR X=0 TO 38
200      C=STATUS(X,Y)¥&H20
210      COLOR C
220      LOCATE X,Y
230      PRINT USING"#";C;
240    NEXT
250 NEXT
```

第16章 ハードウエア仕様

16-1 PC-8801mkII回路図

本回路図は著者ならびにシステムソフト技術部が独自に調査したもので、文責は著者ならびにシステムソフトにかかるものであることをお断わりしておきます。

図-1 CPU

図-2 ROM

図-3 RAM

図-4 DMAC

図-5 GF-1

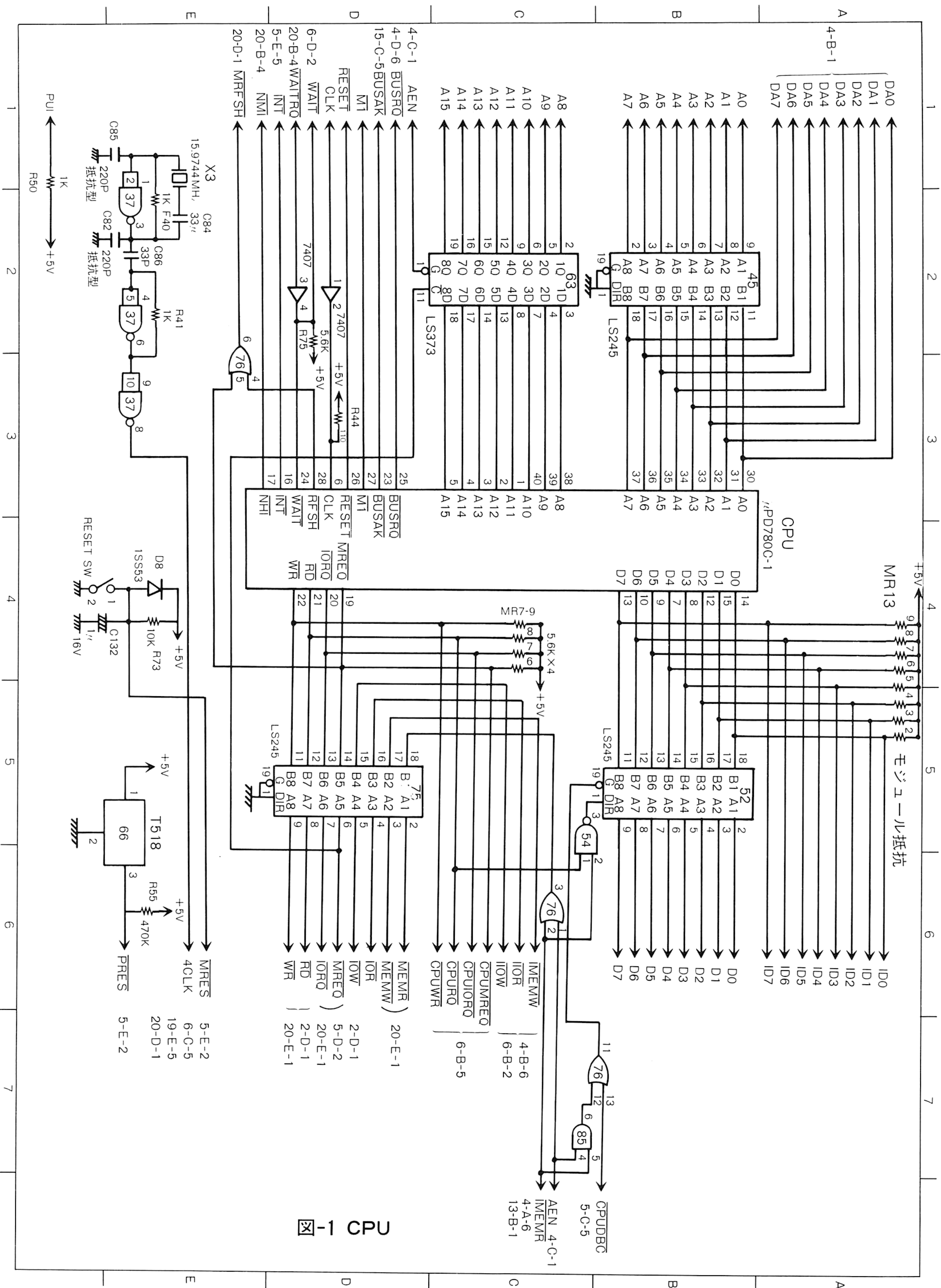
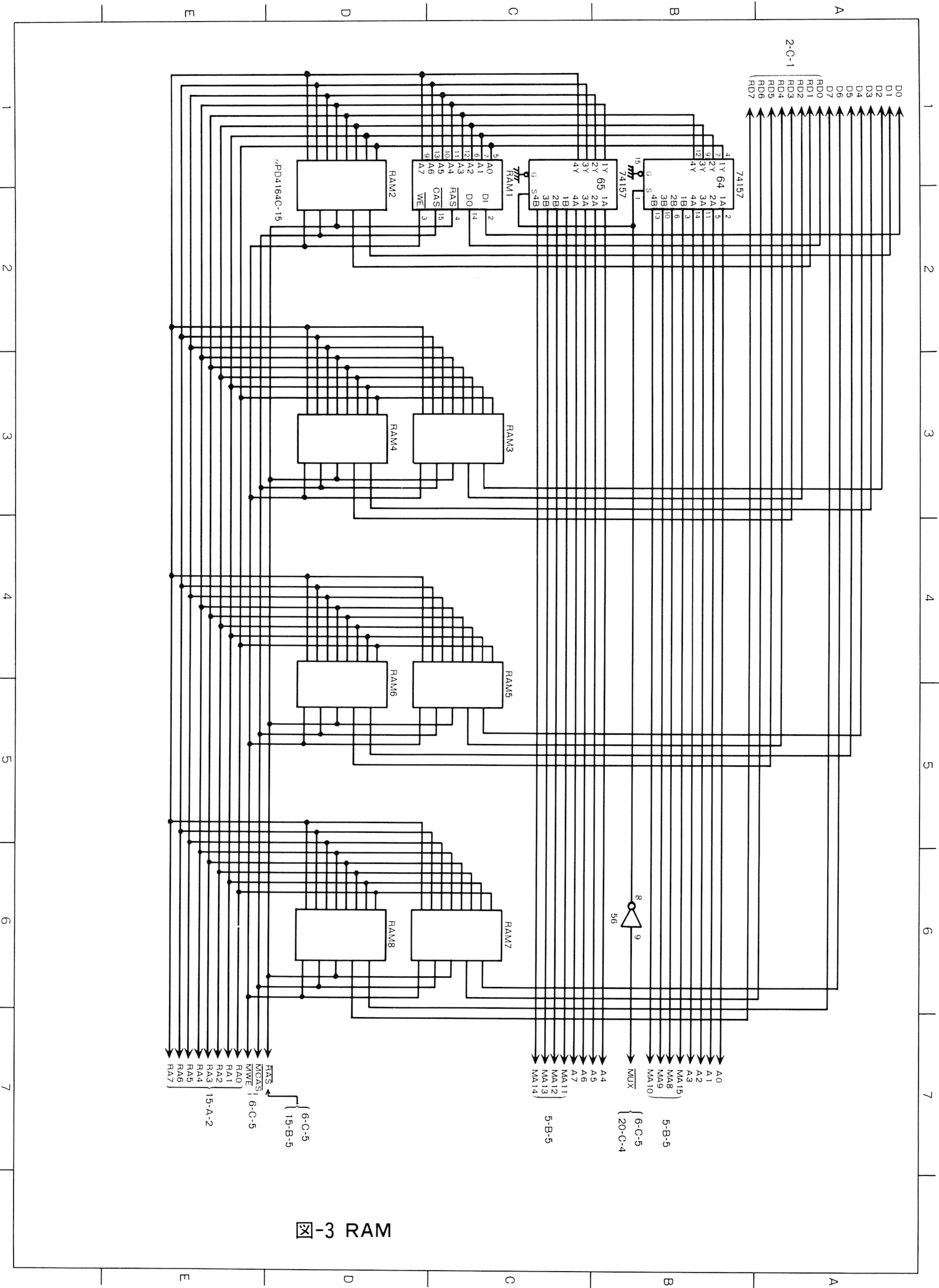
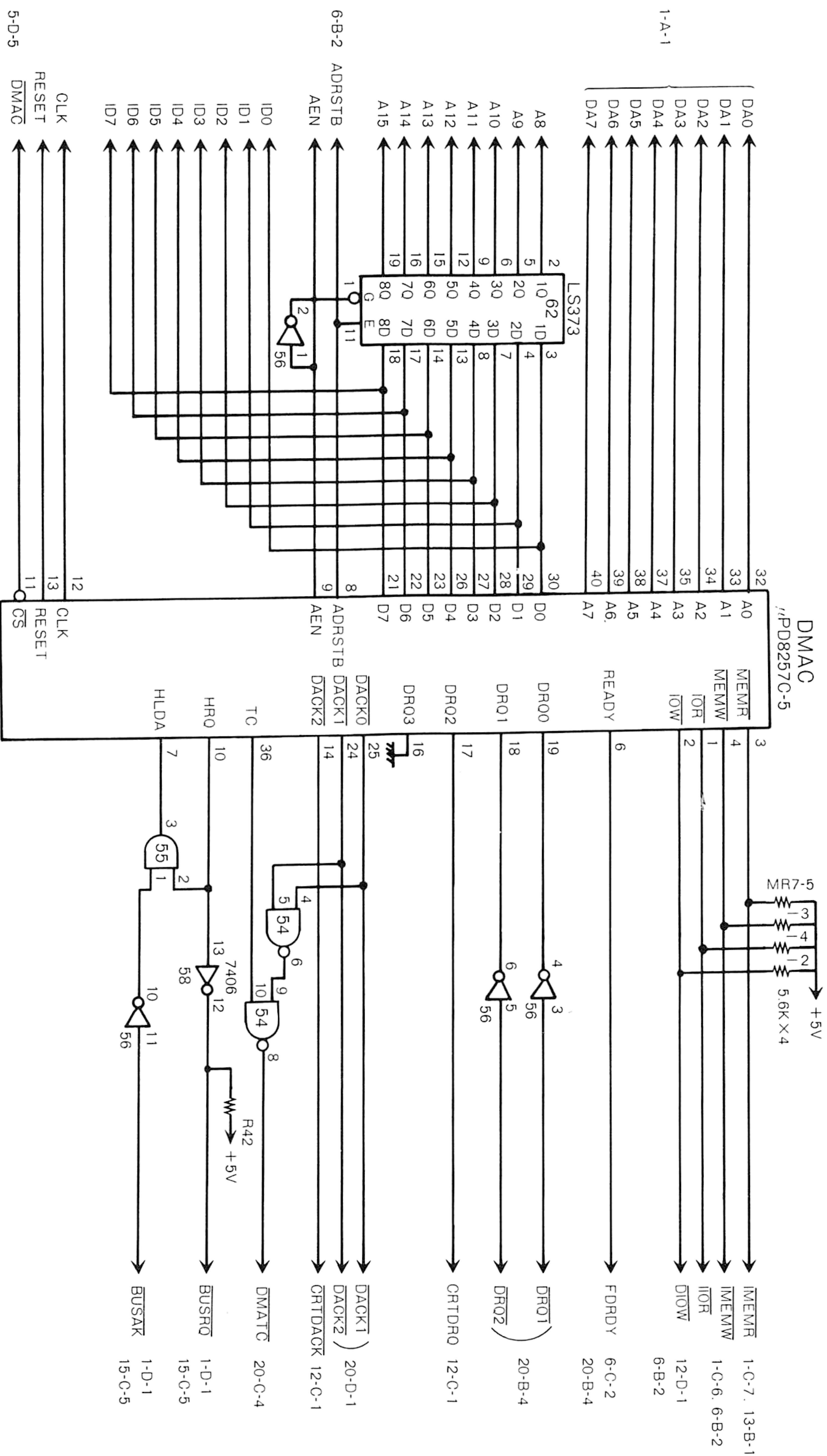
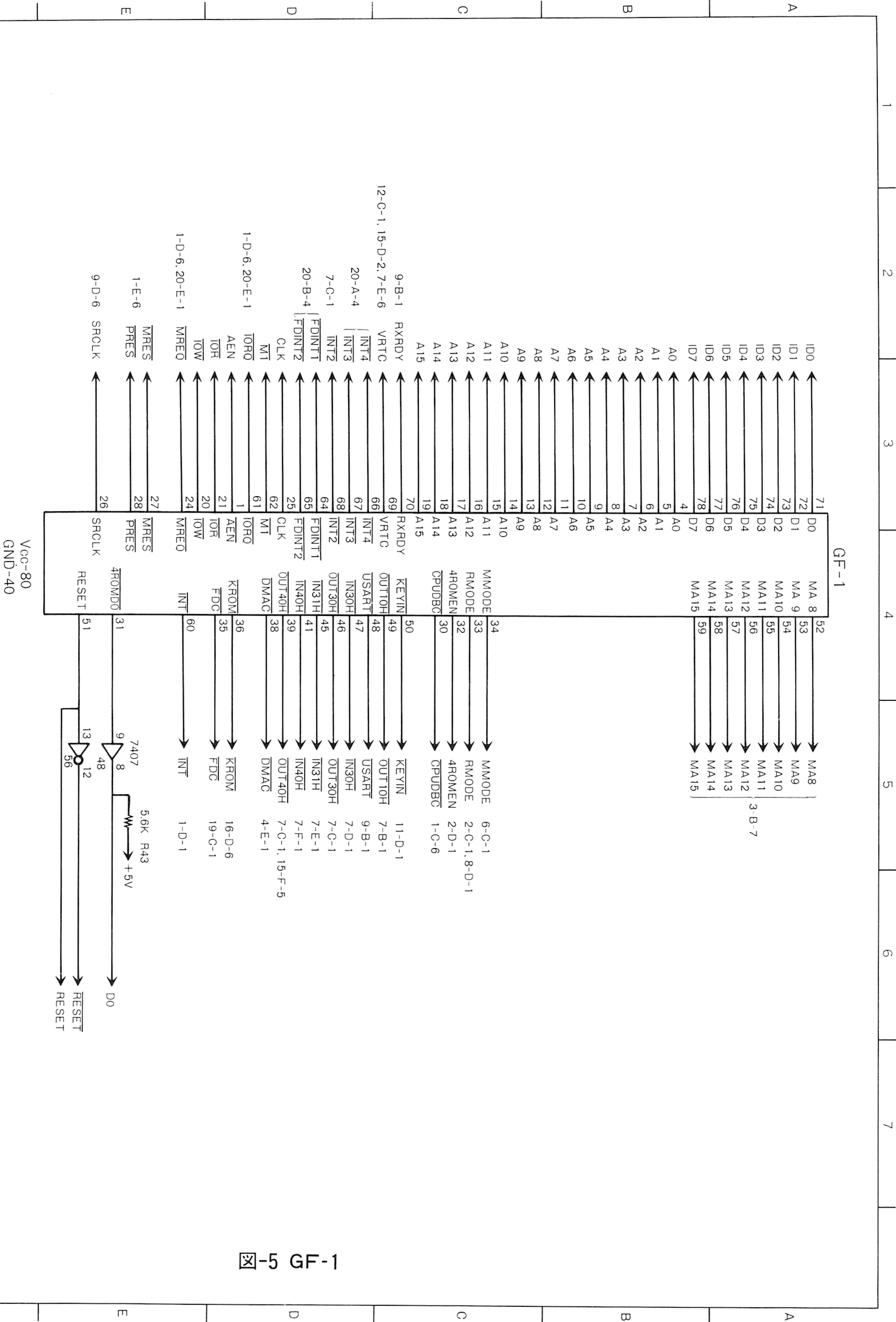



図-6 GF-2

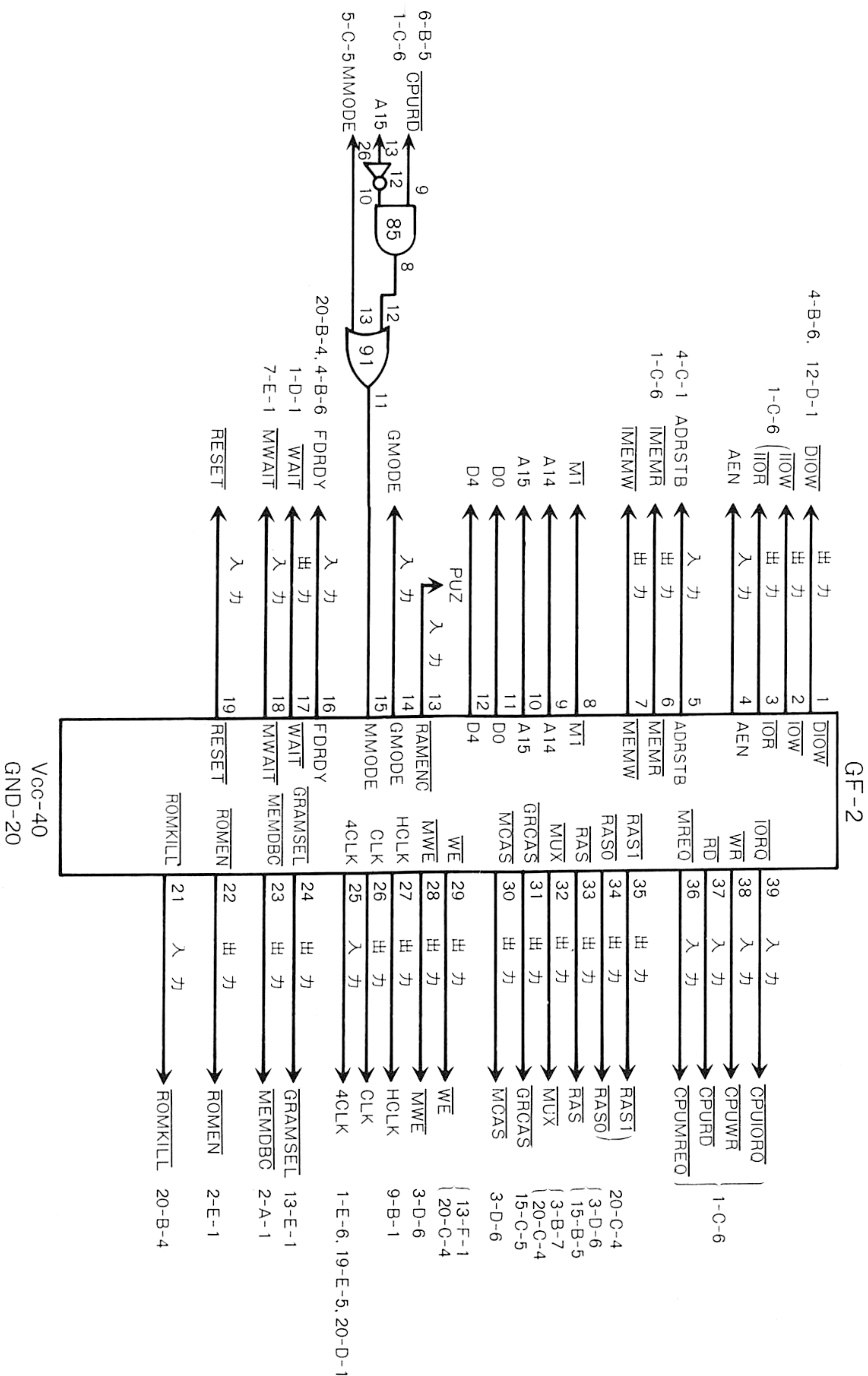
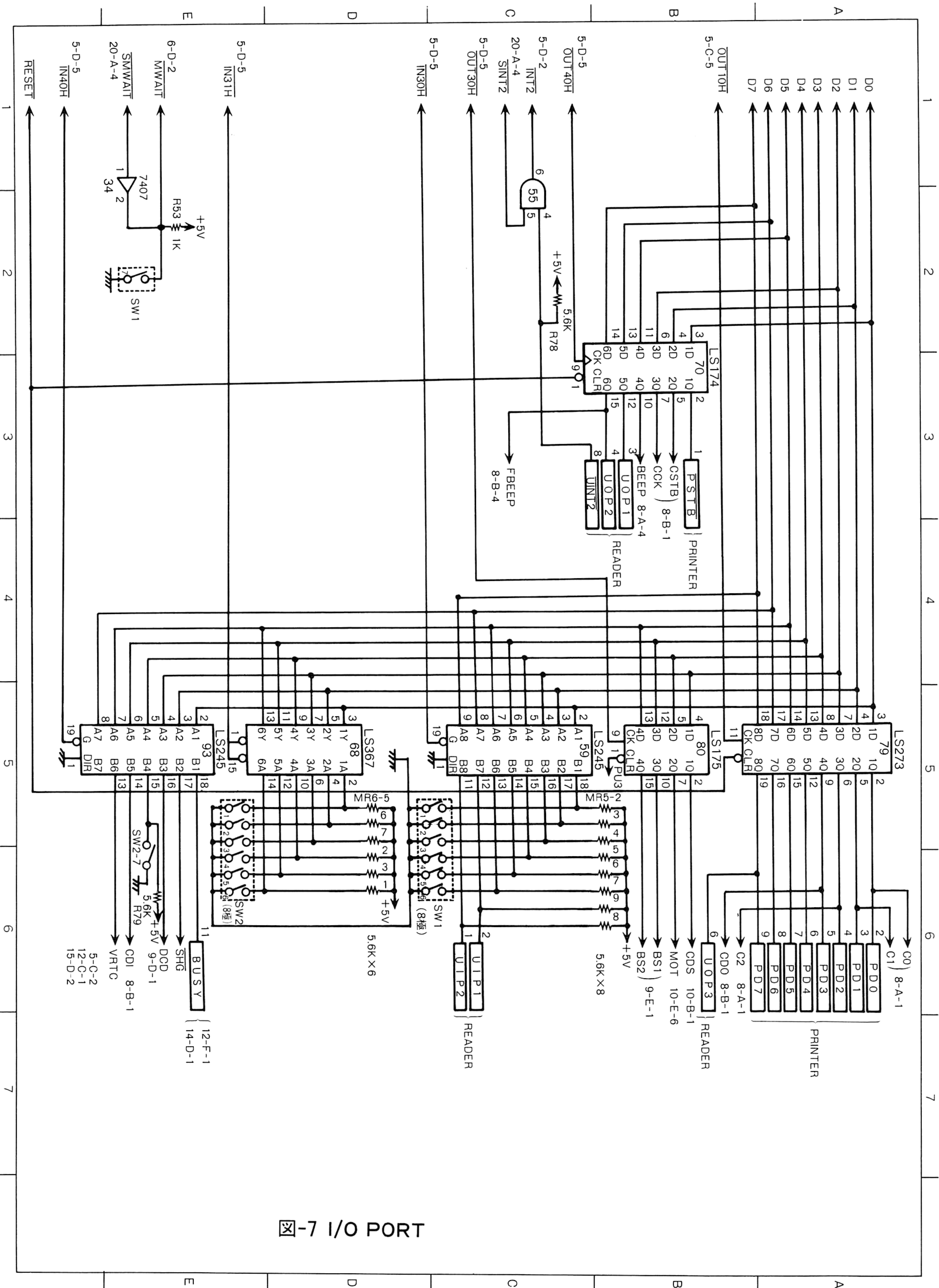



図-7 I/O PORT

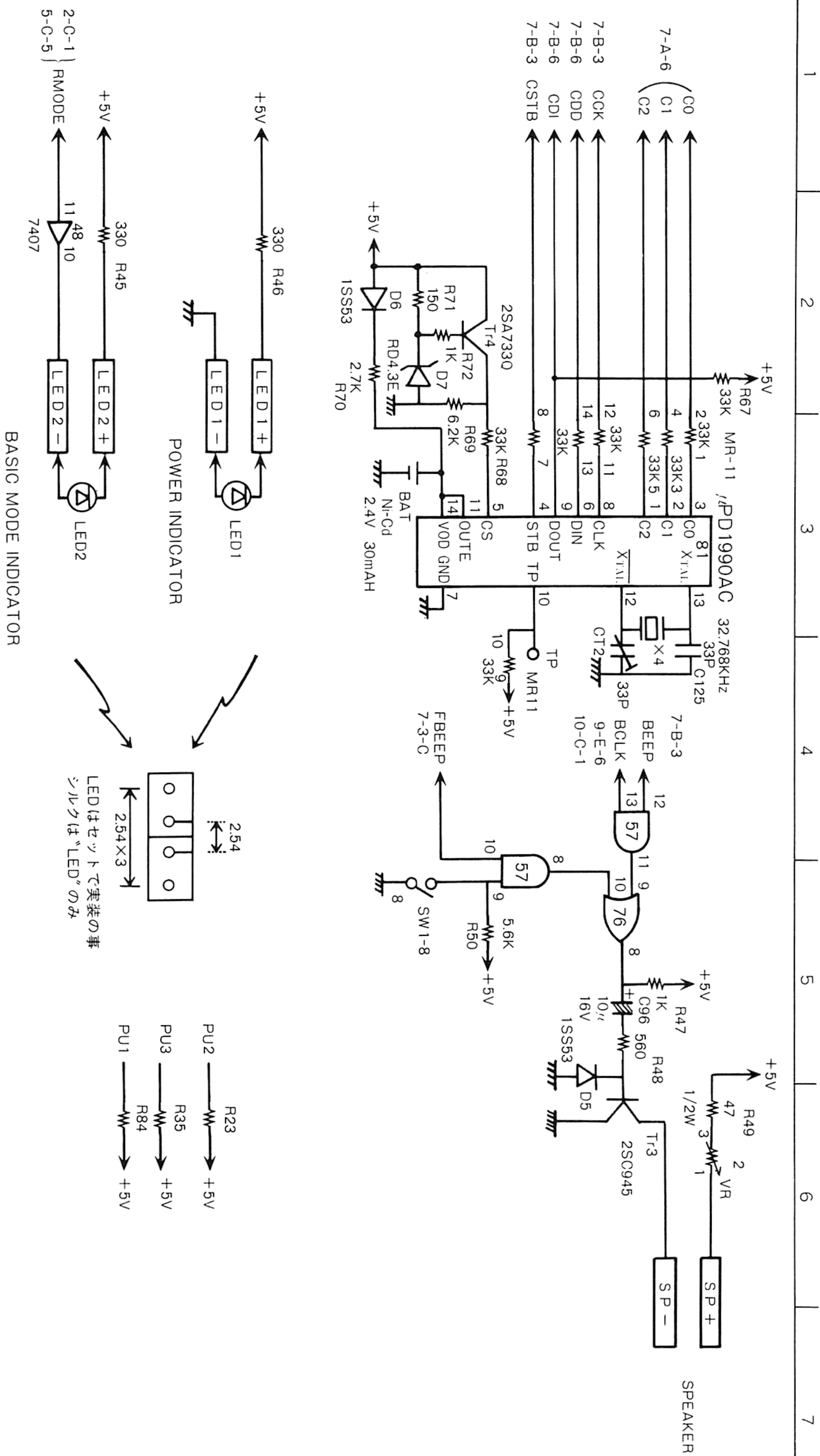



図-8 CLOCK

図-9 シリアルインタフェース

図-11 KEY

図-12 CRTC

図-13 GVAM

図-14 VIDEO

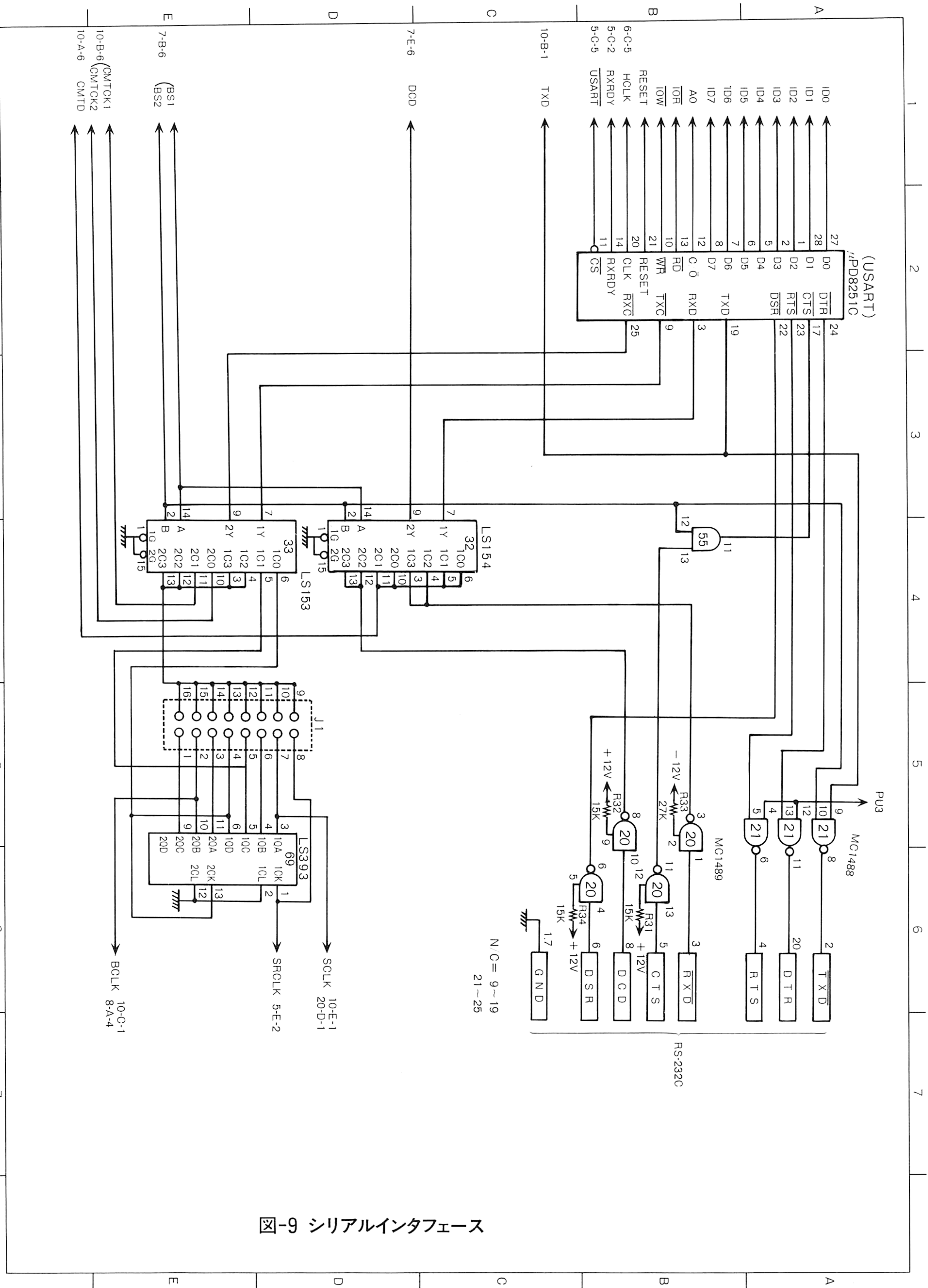
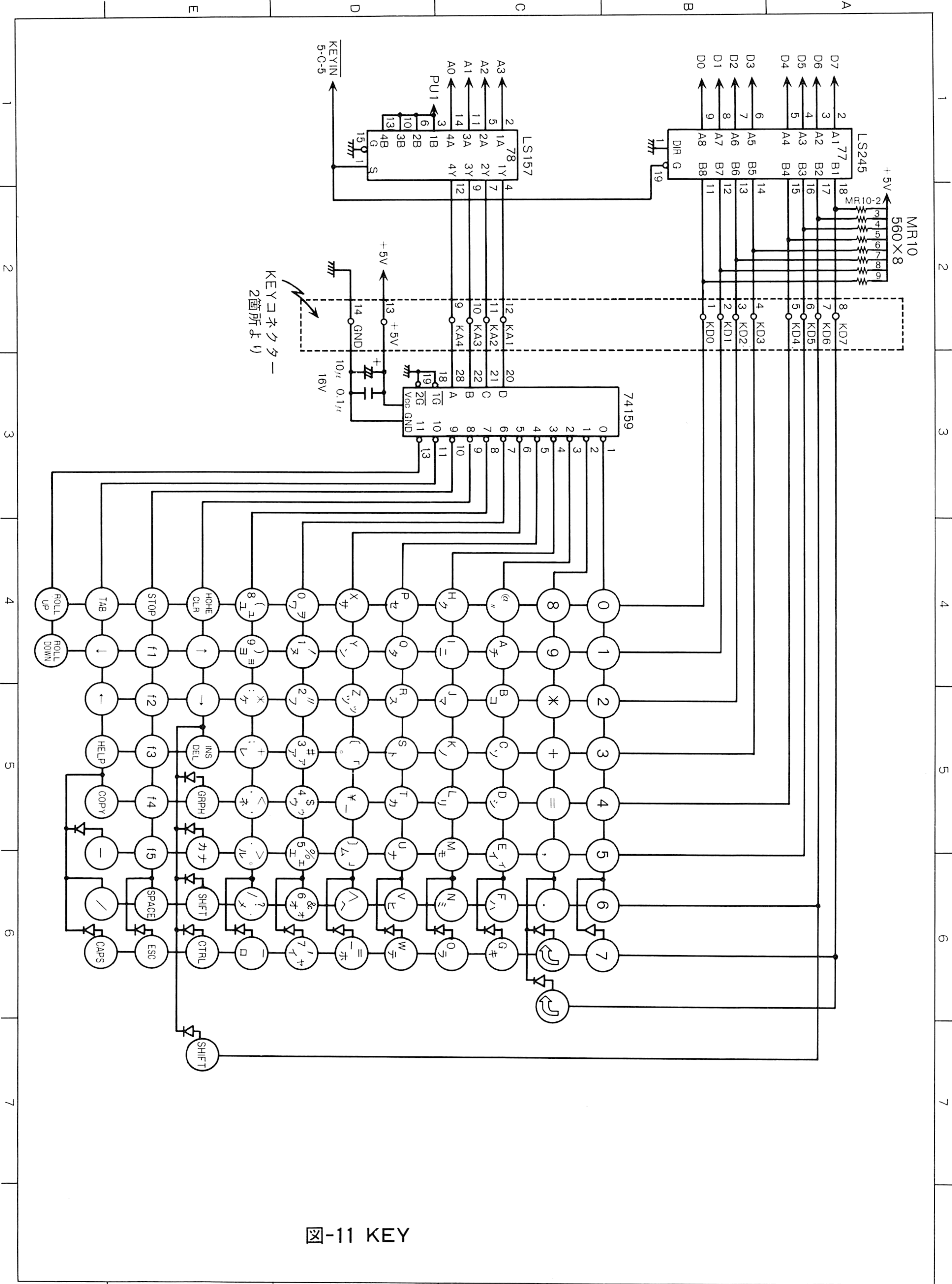
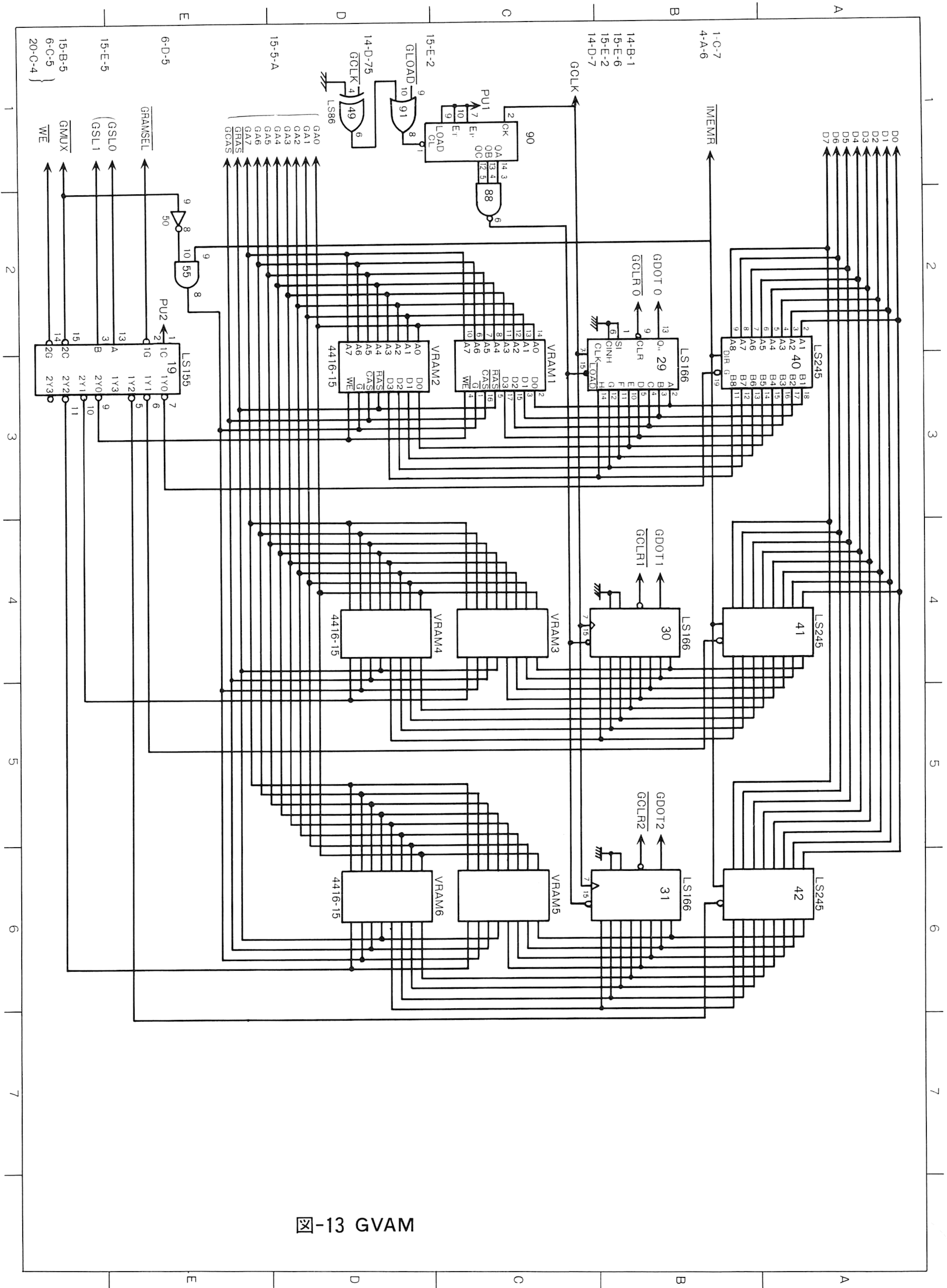
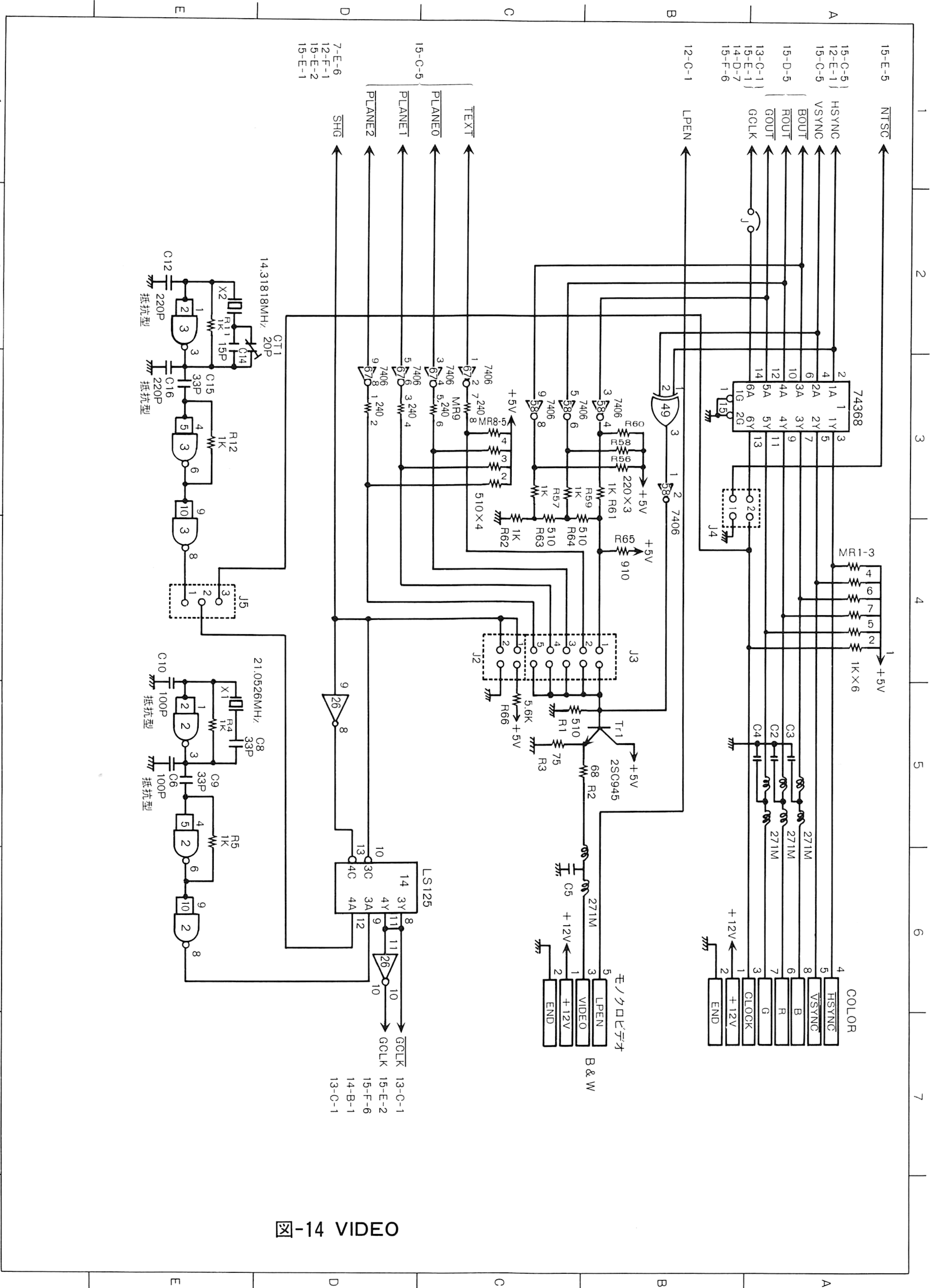
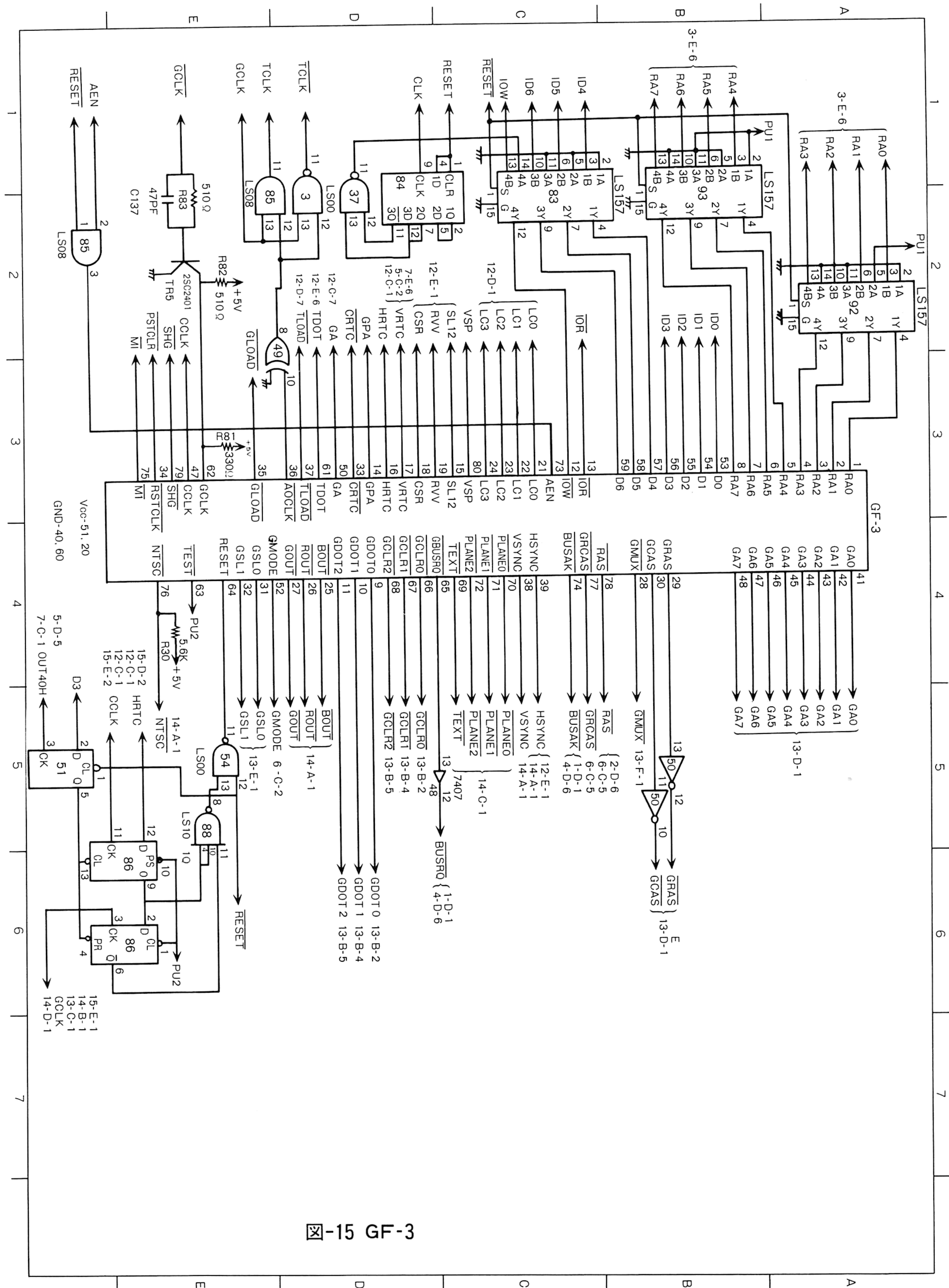
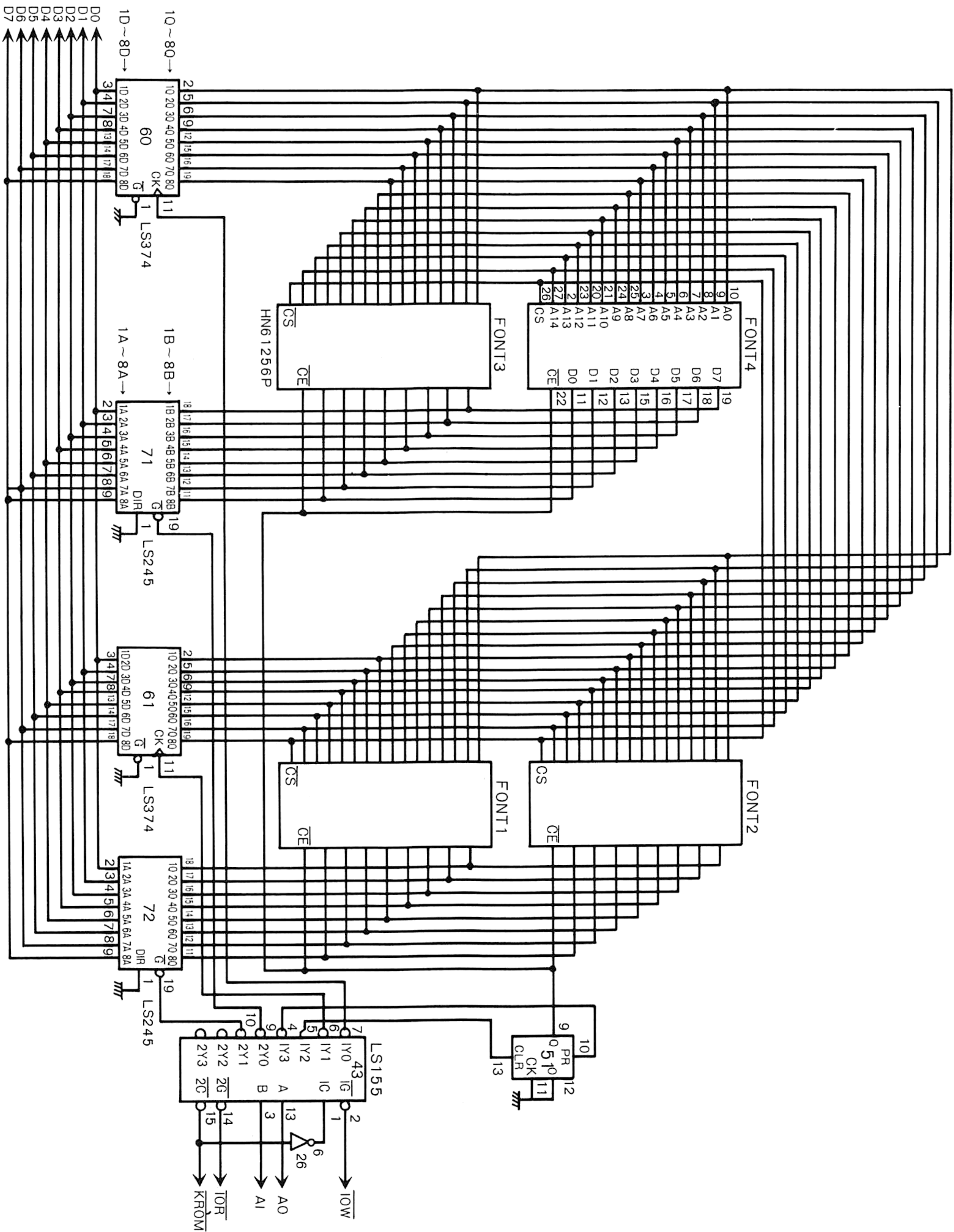
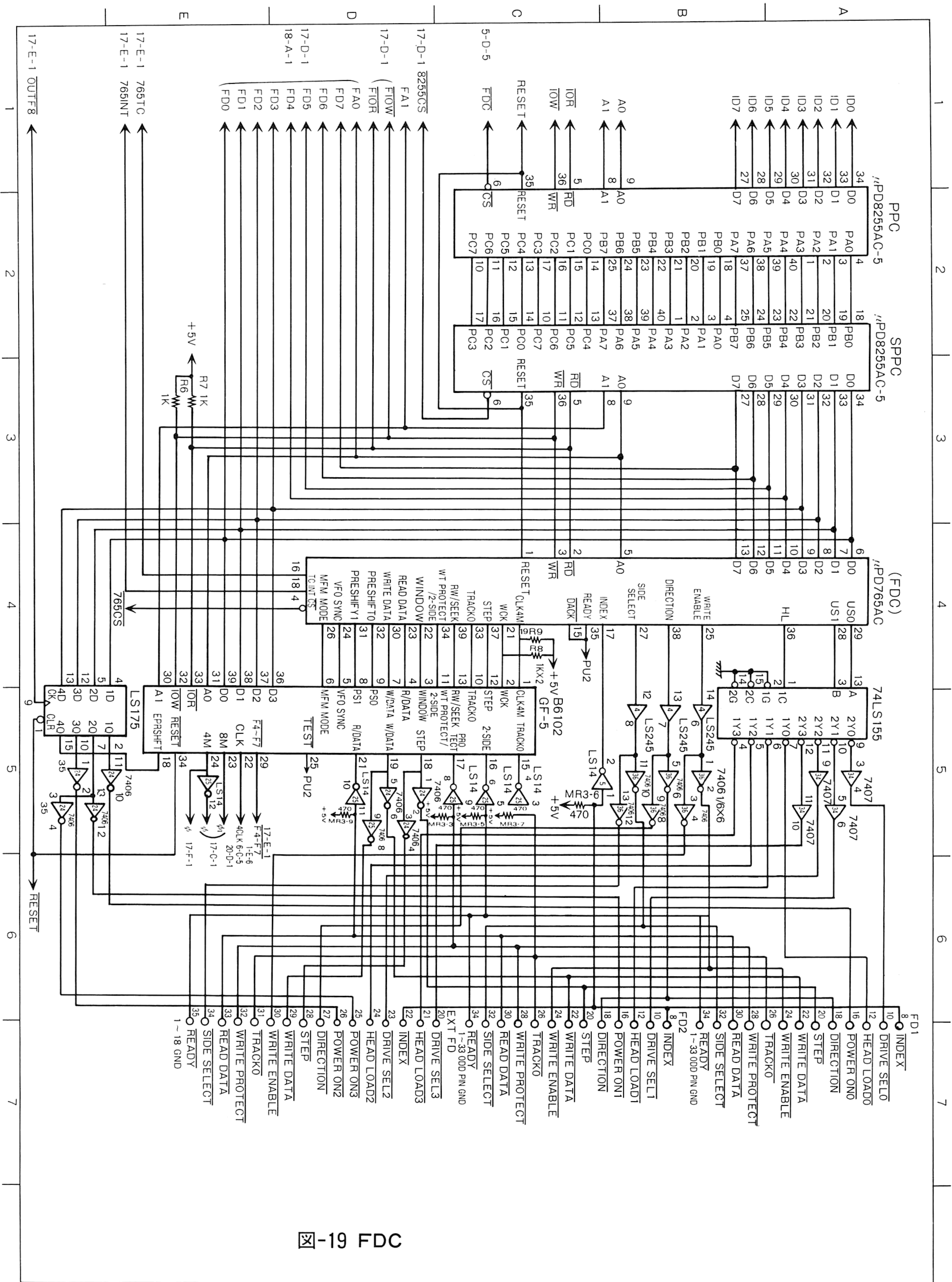
図-15 GF-3

図-16 漢字ROM

図-17 FDC CPU

図-18 FDC RAM

図-19 FDC

図-20 バススロケット

16-2 メインシステムI/Oポート一覧表

ポートアドレス		内容
0 0 H 〜 0 B H	0 〜 1 1	キーボード (IN)

(データ・バス)

(アドレス・バス)	D0	D1	D2	D3	D4	D5	D6	D7
00	0	1	2	3	4	5	6	7
01	8	9	*	+	=	,	.	↙
02	@ ~ ″	A チ	B コ	C ソ	D シ	E イ	F ハ	G キ
03	H ク	I ニ	J マ	K ノ	L リ	M モ	N ミ	O ラ
04	P セ	Q タ	R ス	S ト	T カ	U ナ	V ヒ	W テ
05	X サ	Y ン	Z ッ ツ	[。「	¥ _ |	] ム 」	^ ヘ	_ = ホ
06	0 ヲ ワ	1 ! ヌ	2 ″ フ	3 # ア ァ	4 $ ウ ゥ	5 % エ ェ	6 & オ ォ	7 ' ヤ ャ
07	8 (ユ ュ	9) ヨ ョ	: * ケ	; + レ	, < ネ 、	. > ル 。	/ ? メ ・	_ ロ
08	HOME CLR	↑	→	INS DEL	GRPH	カナ	SHIFT	CTRL
09	STOP	f・1	f・2	f・3	f・4	f・5	SPACE	ESC
0A	TAB	↓	←	HELP	COPY	−	/	CAPS
0B	ROLL UP	ROLL DOWN						

ポートアドレス		内　　　容
10H	16	プリンタ・データ出力ポート（OUT） プリンタ用バスにデータを出力します。 7 6 5 4 3 2 1 0 OUT: PDB7 PDB6 PDB5 PDB4 PDB3 PDB2 PDB1 PDB0
10H	16	カレンダークロック（μPD1990）用出力ポート（OUT） μPD1990にコマンドやデータを出力します。 7 6 5 4 3 2 1 0 OUT: — — — — CDO C2 C1 C0
10H	16	汎用出力ポートUOP 3にデータを出力します。 7 6 5 4 3 2 1 0 UOP3 — — — — — — —

bit		内　　　容
0 1 2	C0 C1 C2	μPD1990へのコマンド出力
3	CDO	μPD1990へのデータ出力

ポートアドレス		内　　　容
2 0 H	3 2	USART（μPD8251）データポート（IN/OUT）
2 1 H	3 3	USART（μPD8251）コントロールポート（IN/OUT）
		モードインストラクションの定義（OUT）

非同期モード

MSB | S_2 | S_1 | EP | PEN | L_2 | L_1 | B_2 | B_1 | LSB

ボーレート

B_1	0	1	0	1
B_2	0	0	1	1
	同期モード		X16	X64

キャラクタ長

L_1	0	1	0	1
L_2	0	0	1	1
	5ビット	6ビット	7ビット	8ビット

PEN → パリティイネーブル
1：イネーブル
0：ディスエイブル

EP → パリティ指定
1：偶　数
0：奇　数

ストップビット数

S_1	0	1	0	1
S_2	0	0	1	1
	無　効	1ビット	$1\frac{1}{2}$ビット	2ビット

ポートアドレス		内　容
		同期モード

SCS	ESC	EP	PEN	L_2	L_1	0	0

キャラクタ長

0	1	0	1
0	0	1	1
5ビット	6ビット	7ビット	8ビット

PEN → パリティイネーブル
1：イネーブル
0：ディスエイブル

EP → パリティ指定
1：偶　数
0：奇　数

ESC → 外部同期検出
1：SYNDET＝入力
0：SYNDET＝出力

SCS → 単一キャラクタ同期
1：単一SYNCキャラクタ
0：ダブルSYNCキャラクタ

コマンドインストラクションの定義（OUT）

EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN

TxEN → 送信イネーブル
1：イネーブル
0：ディスエイブル

DTR → データ・ターミナルレディ
1：Date Terminal ReadyをONにする。
0：Date Terminal ReadyをOFFにする。

RxE → 受信イネーブル
1：イネーブル
0：ディスエイブル

SBRK → センドブレイク・キャラクタ
1：ブレイクキャラクタの送信
0：通常動作

ER → エラーリセット
1：エラーフラグ(PE,OE,FE)をリセット
0：NO OPERATION

RTS → センド要求
1：Request to SendをONにする。
0：Request to SendをOFFにする。

IR → 内部リセット
1：8251をモード・インストラクションフォーマットへもどす。
0：NO OPERATION

EH → HUNT
1：SYNCキャラクタ検出を始める。
0：NO OPERATION

ポートアドレス		内　　　容
		ステータスの読み出し（IN）

DSR	SYN DET	FE	OE	PE	TxE	Rx RDY	Tx RDY

- TxRDY → 送信レディ
  1：レディ
  0：ビジー
- RxRDY → 受信レディ
  1：レディ
  0：ビジー
- TxE → 送信バッファエンプティ
  1：エンプティ
  0：フル
- PE → パリティエラー
  1：パリティエラー発生
  0：エラーなし
- OE → オーバーランエラー
  1：オーバーランエラー発生
  0：エラーなし
- FE → フレミングエラー
  1：フレミングエラー発生
  0：エラーなし
- SYNDET → SYNCキャラクタ検出
  1：SYNCキャラクタ検出
  0：検出なし
- DSR → Data Set Ready
  1：Data Set Ready ON
  0：Data Set Ready OFF

（Data Set Ready端子の状態をモニタできます。）

ポートアドレス		内　　　　　容
３０Ｈ	４８	システムコントロールポート(1)

	7	6	5	4	3	2	1	0
OUT	／	／	BS２	BS１	$\overline{\text{MTON}}$	CDS	$\overline{\text{COLOR}}$	$\overline{\text{40}}$

bit		内　　　　　容
0	$\overline{\text{40}}$	CRTディスプレイFORMATコントロール ０：40文字モード １：80文字モード
1	$\overline{\text{COLOR}}$	CRTディスプレイモードコントロール ０：カラーモード １：モノクロモード
2	CDS	CMTキャリアコントロール ０：スペース １：マーク
3	MTON	CMTのモータコントロール ０：OFF １：ON
4 5	BS２ BS１	USARTのチャンネルコントロール BS２　BS１ ０　０　：CMT600bps ０　１　：CMT1200bps １　０　：RS-232C（非同期） １　１　：RS-232C（同期）

N88-BASICのワークエリア：E6C0H番地

ポートアドレス	内容
	ディップ・スイッチ、汎用入力ポート

	7	6	5	4	3	2	1	0
IN	UIP2	UIP1	SW1-6	SW1-5	SW1-4	SW1-3	SW1-2	SW1-1

bit	内容		
0	SW1-1	0：	N-BASIC
		1：	N_{88}-BASIC
1	SW1-2	0：	ターミナルモード
		1：	BASIC
2	SW1-3	0：	80文字/行
		1：	40文字/行
3	SW1-4	0：	25行/画面
		1：	20行/画面
4	SW1-5	0：	Sパラメータ有効
		1：	Sパラメータ無効
5	SW1-6	0：	DELコードを処理する。
		1：	DELコードを無視する。
6	UIP1	汎用入力ポート	
7	UIP2		

ポートアドレス		内　容
31H	49	システムコントロールポート(2)

	7	6	5	4	3	2	1	0
OUT			25LINE	HCOLOR	GRPH	RMODE	MMODE	200LINE

bit		内　容
0	200LINE	ハイスピードCRTモードにおけるグラフィックモードのコントロール 0：640×400×1 1：640×200×3
1	MMODE	RAMモードのコントロール 0：ROM,RAMモード 1：64KRAMモード
2	RMODE	ROMモードのコントロール 0：N88-BASICモード（ROM3，4） 1：N-BASICモード（ROM1，2）
3	GRPH	グラフィックディスプレイモード 0：グラフィック画面を表示しない。 1：　〃　表示する。
4	HCOLOR	カラーグラフィックディスプレイモード 0：モノクロモード 1：カラーモード
5	25LINE	ハイスピードCRTモードにおけるLINE/FRAMEコントロール 0：20LINEモード 1：25LINEモード

N88-BASICのワークエリア：E6C2H番地

ポートアドレス		内　　容
		ディップ・スイッチ

	7	6	5	4	3	2	1	0
IN			SW2-6	SW2-5	SW2-4	SW2-3	SW2-2	SW2-1

bit	内		容
0	SW2-1	0：	パリティ有り
		1：	パリティ無し
1	SW2-2	0：	偶数(EVEN)パリティ
		1：	奇数(ODD)パリティ
2	SW2-3	0：	8ビット
		1：	7ビット
3	SW2-4	0：	ストップ・ビット＝2
		1：	ストップ・ビット＝1
4	SW2-5	0：	Xパラメータ有効
		1：	Xパラメータ無効
5	SW2-6	0：	半二重
		1：	全二重

ポートアドレス		内　　容
40H	64	ストローブポート

	7	6	5	4	3	2	1	0
OUT	UOP2	UOP1	BEEP	FLASH	$\overline{\text{CLDS}}$	CCK	CSTB	$\overline{\text{PSTB}}$

bit		内　　容
0	$\overline{\text{PSTB}}$	プリンタへのストローブ信号 0：アクティブ 1：インアクティブ
1	CSTB	カレンダクロックへのストローブ信号 0：インアクティブ 1：アクティブ
2	CCK	カレンダクロックへのシフトロック 0：ノーマル 1：クロックON
3	$\overline{\text{CLDS}}$	CRTコントロール回路への同期パルス 0：インアクティブ 1：アクティブ
4	FLASH	フラッシングモードのコントロール 0：ノーマルモード 1：フラッシングモード
5	BEEP	ブザーのコントロール 0：BEEP OFF 1：BEEP ON
6	UOP1	汎用出力ポート
7	UOP2	汎用出力ポート／サウンドポート ディップSW1-8で切り換える

ポートアドレス	内　　　容

	7	6	5	4	3	2	1	0
IN			VRTC	CDI	$\overline{EXTON}$	DCD	$\overline{SHG}$	BUSY

bit		内　　　容
0	BUSY	プリンタからのBUSY信号 0：READY 1：BUSY
1	$\overline{SHG}$	ハイレゾリューショングラフィックモード信号 0：ハイレゾモード 1：ノーマルモード
2	DCD	SIOのデータキャリアディテクト信号 0：キャリア入力なし 1：キャリア入力あり
3	$\overline{EXTON}$	ミニディスクユニット接続信号 0：接続されている　(ディップSW２－７ 1：接続されていない　の状態)
4	CDI	カレンダクロックからのデータ入力
5	VRTC	CRTCからの垂直帰線信号 0：表示，水平帰線サイクル 1：垂直帰線サイクル

ポートアドレス		内　　　容
50H	80	CRTC（μPD3301）パラメータ　（OUT）
51H	81	CRTC（μPD3301）コマンド　（OUT）
52H	82	ボーダーカラー，バックグラウンドカラー制御（OUT）

	7	6	5	4	3	2	1	0
OUT	／	BGG	BGR	BGB	／	RG	RR	RB

bit		内　　　容	
0	RB	ボーダーカラー	BLUE
1	RR	〃	RED
2	RG	〃	GREEN
4	BGB	バックグラウンドカラー	BLUE
5	BGR	〃	RED
6	BGG	〃	GREEN

ポートアドレス		内　　　容
53H	83	画面の重ね合わせの制御（OUT）

	7	6	5	4	3	2	1	0
OUT	／	／	／	／	G2DS	G1DS	G0DS	TXTDS

bit		内　　　容
0	TXTDS	TEXT画面の表示 0：表示する 1：表示しない
1	G0DS	G-VRAM0の表示 0：表示する 1：表示しない
2	G1DS	G-VRAM1の表示 0：表示する 1：表示しない
3	G2DS	G-VRAM2の表示 0：表示する 1：表示しない

ポートアドレス		内容
54H	84	カラーパレット　0　の制御（OUT）
55H	85	カラーパレット　1　の制御（OUT）
56H	86	カラーパレット　2　の制御（OUT）
57H	87	カラーパレット　3　の制御（OUT）
58H	88	カラーパレット　4　の制御（OUT）
59H	89	カラーパレット　5　の制御（OUT）
5AH	90	カラーパレット　6　の制御（OUT）
5BH	91	カラーパレット　7　の制御（OUT）

	7	6	5	4	3	2	1	0
OUT						PG	PR	PB

bit		内容
0	PB	カラーパレット　BLUE
1	PR	カラーパレット　RED
2	PG	カラーパレット　GREEN

ポートアドレス		内容
5CH	91	G VRAMステータス（IN）

	7	6	5	4	3	2	1	0
IN						G2	G1	G0

bit		内容
0	G0	G-VRAM0ステータス
1	G1	G-VRAM1ステータス
2	G2	G-VRAM2ステータス

ポートアドレス		内容
5CH	92	G-VRAM0　選択　（OUT）
5DH	93	G-VRAM1　選択　（OUT）
5EH	94	G-VRAM2　選択　（OUT）
5FH	95	メインRAM　選択　（OUT）

ポートアドレス		内容
60H〜68H	96〜104	DMAC（μPD8257）コントロールポート 60H〜67H：OUT 68H：IN／OUT

PD8257レジスタ選択表

ポートアドレス		レジスタ	バイト	A_3	A_2	A_1	A_0	F/L	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
60H	96	CH-0 DMAアドレス	下位	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
			上位	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
61H	97	CH-0 ターミナルカウント	下位	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
			上位	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
62H	98	CH-1 DMAアドレス	下位	0	0	1	0	0	チャネル0に同じ							
			上位	0	0	1	0	1								
63H	99	CH-1 ターミナルカウント	下位	0	0	1	1	0								
			上位	0	0	1	1	1								
64H	100	CH-2 DMAアドレス	下位	0	1	0	0	0								
			上位	0	1	0	0	1								
65H	101	CH-2 ターミナルカウント	下位	0	1	0	1	0								
			上位	0	1	0	1	1								
66H	102	CH-3 DMAアドレス	下位	0	1	1	0	0								
			上位	0	1	1	0	1								
67H	103	CH-3 ターミナルカウント	下位	0	1	1	1	0								
			上位	0	1	1	1	1								
68H	104	モードセット(プログラム時)	——	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
		ステータス(読み出し時)	——	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

CH・0　　5インチDMAタイプディスクユニット
CH・1　　8インチディスクユニット
CH・2　　CRTC

※A_0〜A_{15}：DMA開始アドレス　　C_0〜C_{13}：TCの値（N－1）
RdとWr：ベリファイ（００），ライト（０１），リード（１０）
AL：オートロード　　TCS：TCストップ
EW：拡張ライト　　RP：回転優先
EN３〜０：チャネルイネーブル　　UP：UPDATEフラグ
TC３〜０：TCステータスビット

ポートアドレス		内容
70H	112	TEXT WINDOWアドレスのオフセットアドレスレジスタ（IN/OUT）

	7	6	5	4	3	2	1	0
OUT IN	AP15	AP14	AP13	AP12	AP11	AP10	AP9	AP8

ポートアドレス		内　　　容
71H	113	4th ROMコントロール 4th ROMを制御します。

	7	6	5	4	3	2	1	0
OUT IN	$\overline{\text{4th}}$ ROM8	$\overline{\text{4th}}$ ROM7	$\overline{\text{4th}}$ ROM6	$\overline{\text{4th}}$ ROM5	$\overline{\text{4th}}$ ROM4	$\overline{\text{4th}}$ ROM3	$\overline{\text{4th}}$ ROM2	$\overline{\text{4th}}$ ROM1

ポートアドレス		内　　　容
78H	120	TEXT WINDOWアドレスのオフセットアドレスレジスタを1増す。
E4H	228	割込みコントローラ(μPD8214) カレントステータス出力ポート(OUT)

	7	6	5	4	3	2	1	0
OUT	／	／	／	／	$\overline{\text{SGS}}$	$\overline{B_2}$	$\overline{B_1}$	$\overline{B_0}$

bit		内　　　容
0	$\overline{B_0}$	カレントインタラプトレベル
1	$\overline{B_1}$	
2	$\overline{B_2}$	
3	$\overline{\text{SGS}}$	カレントステータスレジスタのコントロール

N88-BASICのワークエリア：E6C3H番地

ポートアドレス		内　　　容
E6H	230	割込みマスクフラグ (OUT)

	7	6	5	4	3	2	1	0
OUT	／	／	／	／	／	$\overline{\text{RXMF}}$	$\overline{\text{VRMF}}$	$\overline{\text{RTMF}}$

bit		内　　　容
0	$\overline{\text{RTMF}}$	リアルタイム割込みのマスクフラグ(0：禁止, 1：許可)
1	$\overline{\text{VRMF}}$	VRTC割込みのマスクフラグ　(0：禁止, 1：許可)
2	$\overline{\text{RXMF}}$	RxRDY割込みのマスクフラグ　(0：禁止, 1：許可)

N88-BASIKのワークエリア：EFOEH番地

ポートアドレス		内　　　容
E8H	232	(OUT) 漢字ROMアドレスの下位8bit指定 (IN) 漢字フォントの読み出し (下位8bit)
E9H	233	(OUT) 漢字ROMアドレスの上位8bit指定 (IN) 漢字フォントの読み出し (上位8bit)
EAH	234	(OUT) 漢字ROMの読み出し開始
EBH	235	(OUT) 漢字ROMの読み出し終了

ポートアドレス		内　　　容
F3H	243	DMAタイプ ディスクユニット インターフェイスセレクトポート(OUT)
F4H ∫ F7H	244 247	DMAタイプ 8インチディスクユニット制御ポート
F8H ∫ FBH	248 251	DMAタイプ 5インチディスクユニット制御ポート

F4H	F8H	インターフェースボードチェック (IN) bit0 モーターコントロール(OUT) PRE-COMPENSATIONのため
F5H	F9H	マージンコントロール (OUT)
F6H	FAH	FDCステータス (IN)
F7H	FBH	FDCデータ・レジスタ(IN/OUT)

FCH ∫ FFH	252 255	ミニディスクユニット制御ポート（μPD8255） FCH：PORT A（IN）入力データ FDH：PORT B（OUT）出力データ FEH：PORT C（IN/OUT）ハンドシェイク FFH：コントロールポート（OUT）

16-3 サブシステムI/Oポート一覧表

ポート・アドレス		内　　容
F6H		［OUT］ プリンタストローブ信号を出力 F6HをOUTするとBUSY信号と同じレベルが出力される。 （PC-8801mkII，PC-80S31では無意味） BUSY / PSTB / DATA IN F7Hで"Low"をチェック（Bit 0が0） プリンタがBUSYにする プリンタがReadyにする OUT F6HでPSTBにBUSYのレベル（Low）がでる。 1 μsec（最低） OUT F6HでPSTBにBUSYのレベル（High）がでる。 OUT F7Hでデータをセット 最低1μsec　最　低　1 μsec　最低1 μsec （数値はPC-PR201参照）
F7H		［OUT］ 1）VFO用ウィンドウ値の出力に下位4ビットを使用 2）プリンタへのデータ出力（8ビット）（PC-8801mkII，PC-80S31では無意味） ［IN］　プリンタのBUSY信号の入力 Bit 0 ＝ { 0　Ready / 1　Busy }
F8H		［OUT］ 1）モーターのON，OFF ビット 7：1　6：1　5：1　4：1　3：ドライブ#3　2：ドライブ#2　1：ドライブ#1　0：ドライブ#0 （1……ON　0……OFF） 2）プリコンペイセイションのセット・リセット 07H —— セット　（0～19トラック） 0FH —— リセット（20～39トラック）
FAH		［IN］　FDC765からステータス・レジスタを入力
FBH		［OUT］［IN］ FDC765へデータを出力・入力する。
FCH ～ FFH		メインシステムとのインターフェイス 8255の制御ポート FCH：ポートA（IN）　入力データ FDH：ポートB（OUT）　出力データ FEH：ポートC（IN/OUT）ハンドシェイク FFH：コントロール・ポート（OUT）

第17章　ソフトウェア解析

17-1-1 インタプリタ

A） N₈₈-BASIC ROM（アドレス0000～7FFF）

アドレス	項目名	機能
0000	リセット	
0008	シンタックスチェック	RST (or CALL) の後に置かれているキャラクタと (HL) とを比較して、一致しなかったら Syntax error、一致したら RST 10H をしてもどる。
0010	テキストからの 1文字読み込み	テキストから1文字 or 数を読み込む。
0018	1文字出力	CRT または LPT に1文字出力をする。Acc に文字コードを入れて RST 18H を行なう。 (E64C) = 0 CRT に出力 ≠0 LPT に出力
0020	HL, DE の比較	HL-DE（無符号）を行ないフラグをセットする。Acc はこわれる。
0028	SGN (FAC)	FAC（単精度，倍精度）の符号を調べる。(cf. 20BD) －：Acc=FF, NZ, M 0：Acc=0, Z, P ＋：Acc=1, NZ, P
0030	FAC の型チェック	FAC の型を調べる。 INT：Acc=FF, C, NZ, M, PE STR：Acc=00, C, Z, P, PE SNG：Acc=01, C, NZ, P, PO DBL：Acc=05, NC, NZ, P, PE
0038	ユーザー用 RST	モニタではブレークポイント用またはコマンド待ちに戻る時に使う。
004A	ダイレクトモードにセット	(E656, 57) に FFFF を入れる。
008A	演算子優先順位テーブル	+, −, *, ／, ^, AND, OR, XOR, EQV, IMP, MOD, ￥の順に優先度を表わす数が入っている。(数字が大きいほど、優先度が高い)
0096	FAC型変換ルーチンアドレステーブル	0096：to DBL 009A：to INT 009C：to STR 009E：to SNG
00A0	倍精度演算ルーチンアドレステーブル	+, −, *, ／, 比較の順で処理ルーチンのアドレスが入っている。 FAC ○ (EC4A～EC5A) →FAC
00AA	単精度演算ルーチンアドレステーブル	BCDE ○ FAC →FAC
00B4	整数演算ルーチンアドレステーブル	DE ○ HL →FAC オーバーフローがなければ HL にも入る。
00BE	ワークエリア初期データ	E600～に転送される。
030E	文字列	DB 'Error', 0
0315	文字列	DB 'in', 0
0319	文字列	DB 'Ok', FF, 0D, 0A, 0
0320	文字列	DB 'Break', 0
0326	インタラプトベクトル初期データ	F300～に転送される。 （2バイト×16組）
0346	スタックサーチ	FOR, GOSUB用のスタックを見つける。
036F	コマンド待ちへ	スタックを文実行前にもどしてコマンド待ちへ。
0375	プログラムエンド処理	プログラムエンドに来た時の処理ルーチン。
0393		Syntax error 出力
0396		Duplicate label 出力
0399		Undefined label 出力
039C		Devision by zero 出力

アドレス	項　目　名	機　　　　能
039F		Next without For 出力
03A2		Duplicate Definition 出力
03A5		Undefined User function 出力
03A8		Resume without error 出力
03AB		Overflow 出力
03AE		Missing operand 出力
03B1		Type mismatch 出力
03B3	エラー出力	Eregにエラーコードを入れてここにジャンプすると,そのエラーが出力される。
047B	Ok出力	Okを表示しコマンド待ちになる。
04A7	コマンド待ち	コマンド待ちになる。
05BD	リンクポインタセット	テキストのリンクポインタを正しくセットする。
05E9	ペア行番号GET	ペアの行番号を読み込む。
0605	行サーチ	テキスト中から行番号＝DE の文を探す。
0632	中間言語変換	(HL)～00のテキストを中間言語に変換し、E87A～に入れる。
08BF	FOR	FOR 文エントリ
0996	次の文を実行	
09C4	次の文を実行	
09E5	1文実行	テキストポインタ＝HL
0A0D	1文字読み込み	RST 10H とほぼ同じ。
0A8D	数→FAC	RST 10H 用FAC→演算用FAC
0AC4	DEFSTR	DEFSTR 文エントリ
0AC7	DEFINT	DEFINT 文エントリ
0ACA	DEFSNG	DEFSNG 文エントリ
0ACD	DEFDBL	DEFDBL 文エントリ
0B01	正の整数式の評価	正の整数式を評価してDEに入れる。
0B06		Illeagal function call 出力
0B0B	行番号読み込み	行番号を読んでDEに入れる。
0B34	行位置読み込み	行番号 or 行アドレスを読み込んでDEに入れる。
0B7C	RUN	RUN エントリ
0BBF	GOSUB	GOSUB 文エントリ
0BE0	割り込み GOSUB	ON 割り込み発生時の GOSUB
0BF9	GOTO	GOTO 文エントリ
0C41	RETURN	RETURN 文エントリ
0C77	DATA	DATA 文エントリ
0C79	REM ELSE	REM　文エントリ ELSE 文エントリ
0C97	FAC→変数	FAC の値を変数にコピーする。 [CALLの方法] Acc ←型－3 DE ←コピー先アドレス PUSH　リターンアドレス PUSH　DE PUSH　AF JP　0C97
0C9C	LET	LET 文エントリ
0D01	ON	ON 文エントリ
0D05	ON ERROR	ON ERROR GOTO 処理
0D39	ON XX GOSUB	ON＜割り込み＞GOSUB（　）の処理
0D73	ON＜式＞GOTO GOSUB	ON～GOTO, GOSUB 処理
0D8D	RESUME	RESUME 文エントリ

アドレス	項　目　名	機　　　能
0DCA	ERROR	ERROR 文エントリ
0DD5	AUTO	AUTO 文エントリ
0E05	IF	IF 文エントリ
0E4C	LPRINT	LPRINT 文エントリ
0E54	PRINT	PRINT 文エントリ
0E8F	文字列出力	FAC の文字列を位置制御して出力する処理を行なっている。
0ED2	コンマ処理	PRINT 文での「,」の処理
0F1B	TAB, SPC 処理	PRINT 文での TAB, SPC の処理
0F8B	カセット OFF	カセットアクセスを終わる。
0FAA	LINE	LINE 文エントリ
0FAF 102D	LINE INPUT INPUT	LINE INPUT 処理 INPUT 文エントリ
1040	プロンプト文処理	(LINE) INPUT のプロンプト文の処理ルーチン。
10F9	READ	READ 文エントリ
11D3	式の評価(FRMEVL)	式を計算して，結果を FAC に入れる。
1341	整数除算（1）	HL/DE→FAC
1350	因子の評価	演算の因子を評価して FAC に入れる。
1394	ERR	ERR 変数→FAC
1398	Acc to FAC	Acc（0～255）を整数型 FAC に入れる。
13A2	ERL	ERL 変数→FAC
13B0	VARPTR	VARPTR→FAC
13F2	（式）の評価	（　）で囲まれた式を評価して FAC に入れる。
1406	変数読み出し	変数の値→FAC
1415	大文字変換	Acc 内が英小文字だったら大文字に変換する。
1512	NOT	NOT 演算子の処理
1581	LPOS	LPOS→FAC（LPOSの値＝E64B 番地）
1586	POS	POS (X) →FAC
158F	USR	USR エントリ
15C8	DEF USR	DEF USR 処理
15D7	DEF	DEF エントリ
1600	FN	FN　エントリ
1796	FAC 型変換	FAC を Acc で示す型に変換する。 Acc＝2　INT 3　STR 4　SNG 8　DBL
17A5	ブロック転送	DE の指す所から HL の指す所へ BC 個（≠0）移動する。
17AF	プログラム実行中 チェック	ダイレクトモード中だったら Illeagal direct エラー
17C9	FF グループ ステートメント	FF グループのステートメントの各処理ルーチンへ分岐する。
17E5	INP	INP (FAC)→FAC
17FA	OUT	OUT 文エントリ
1800	WAIT	WAIT 文エントリ
181A	WIDTH	WIDTH 文エントリ
1856	WIDTH#	Acc ←ファイル#，E ←サイズにして CALL
186A	WIDTH LPRINT	Acc ←文字数にして CALL
1880	WIDTH PRINT	PRINT 文での WIDTH 設定　Acc ←文字数 & CALL
1896	整数式の評価	HL ←テキストポインタ & CALL FAC と DE に値が入る。
18A3	パラメータの評価	値が 0～255となる式の評価．HL ←テキストポインタ．

アドレス	項　目　名	機　　　　能
18A3	パラメータの評価	結果は Acc, Ereg, FAC に入る。
18D4	LLIST	LLIST エントリ
18D9	LIST	LIST エントリ
194C	LIST 形式変換	(HL)～00の中間言語を LIST 形式に変換して入力バッファ(E9B9～)に入れる。
1B40	DELETE	DELETE エントリ
1B7A	PEEK	PEEK (FAC)→FAC
1B84	POKE	POKE エントリ
1B93	アドレスゲット	テキストからアドレス(-32768～65535)を読んで HL に入れる。
1B9D	FAC アドレス変換	FACに入っている-32768～65535の数値を整数に変換し、HLに入れる。
1BBA 1BBB	行番号→アドレス アドレス→行番号	プログラム中の全部の GOTO, GOSUB などの行番号の行番号←→行アドレス変換を行なう。
1C89	OPTION BASE	OPTION エントリ
1CCD	コンソール出力	Acc の文字をコンソールへ出力する。59A4と同じ。 Acc ←CHR コード & CALL
1CD1	RANDOMIZE	RANDOMIZE エントリ
1D2A	ループエンドサーチ	Creg＝1Aのとき、NEXT サーチ ≠1Aのとき、WEND サーチ
1DDB	SNG FAC インクリメント	単精度 FAC をインクリメントする。
1DE6	単精度減算	BCDE－FAC→FAC
1DE9	単精度加算	BCDE＋FAC→FAC
1F10	LOG	LOG (FAC)→FAC
1F53	単精度乗算	BCDE＊FAC→FAC
1FB7	単精度除算	BCDE/FAC→FAC
20A0	ABS	ABC (FAC) →FAC
20AB	NEG (FAC) (1)	単精度，倍精度 FAC の符号反転。
20B3	SGN	SGN (FAC) →FAC
20B6	Acc →FAC	Acc (－128～127)を整数型 FAC に入れる。
20BD	SGN(FAC)	FAC の符号を調べる。 ＋：AF＝01, NZ, P 0：　00,　Z, P －：　FF, NZ, M
20CD	単精度 FAC PUSH	単精度の FAC を PUSH する。
20DA		(HL)～(HL＋3)→SNG FAC
20DD		BCDE→SNG FAC
20E8		SNG FAC →BCDE
20EB		(HL)～(HL＋3) →E, D, C, B
20ED		(HL)～(HL＋2) →D, C, B
20F4		SNG FAC →(HL)～(HL＋3)
2107	フォーマット変換	FAC, BCDE 記憶用フォーマット→演算用フォーマット
212B	FAC アドレスゲット	DE←タイプ別 FAC アドレス
2134 215F 2199	単精度比較 整数比較 倍精度比較	BCDE－FAC を行なって Acc, フラグをセットする。 DE－HL を行なって Acc, フラグをセットする。 FAC－(EC4A～EC51)を行なってAcc、フラグをセットする。
21A0	CINT	FAC を整数型に変換する。
21FD	整数生成	HL → INT FAC
2214	CSNG	FAC を単精度に変換する。
223E	CDBL	FAC を倍精度に変換する。
2256	ストリングチェック	FAC が文字型であることのチェック。(違うと Type mismatch)
2286	FIX	FIX (FAC) → FAC

アドレス	項　目　名	機　　　　　能
2295	INT	INT (FAC)→FAC
231F	整数減算	DE−HL →FAC, HL
233A	整数加算	DE+HL →FAC, HL
235A	整数乗算	DE*HL →FAC, HL
23AB	整数除算（2）	DE¥HL →FAC, HL
23F7	NEG (FAC)（2）	整数型 FAC の符号反転
240C	整数剰余	DE mod HL →FAC, HL
241D	倍精度減算	FAC−（EC4A〜EC51）→FAC
2424	倍精度加算	FAC+（EC4A〜EC51）→FAC
2553	倍精度乗算	FAC*（EC4A〜EC51）→FAC
2629	倍精度除算	FAC／（EC4A〜EC51）→FAC
26B5	倍精度数入力	数値（外部形式、ポインタ HL）→FAC
26BC	単精度数入力	数値（外部形式、ポインタ HL）→FAC
28D0	浮動小数点出力 (free format)	FAC →EC53〜に数字（外部形式）
28D1	浮動小数点出力 (fixed format)	FAC →EC53〜に数字（外部形式） [パラメータ] Breg←整数部の桁数（小数点は含まない） Creg←小数部の桁数（　〃　） Areg　bit 7：1 bit 6：3ケタごとの区切り「,」を入れる／入れない bit 5：先頭を「*」で埋める／埋めない bit 4：$をつける／つけない bit 3：+をつける／つけない bit 2：符号は数字の後/前 bit 1：未使用 bit 0：指数形式
2E05	SQR（単精度）	SQR (FAC)→SQR
2E15	べき乗（単精度）	BCDE^FAC→FAC
2E6E	EXP（単精度）	EXP (FAC)→FAC
2ED8	多項式計算（1） （単精度）	C0*FAC+C1*FAC3+C2*FAC5+　…→FAC
2EE7	多項式計算（2） （単精度）	C0+C1*FAC+C2*FAC2+C3*FAC3+…→FAC
2F1A	RND（単精度）	RND (FAC)→FAC
2F8B	COS（単精度）	COS (FAC)→FAC
2F91	SIN（単精度）	SIN (FAC)→FAC
302C	TAN（単精度）	TAN (FAC)→FAC
3041	パラメータテーブル 1	DMA タイプ　5インチディスク用パラメータ
3053	パラメータテーブル 2	DMA タイプ　8インチディスク用パラメータ
3065	シフトテーブル	シフト用データ
306A		ビットを取り出すマスク（右シフト用）
3072		ビットを取り出すマスク（左シフト用）
3079		未使用インタラプト用ルーチン。何もせずに戻る。
3080	VRTC インタラプト	VRTC インタラプト・ルーチン PORT E6H, bit 1 ←0　でマスクできる
3167	RS-232C インタラプト	RS-232C 入力インタラプト・ルーチン PORT E6H, bit 2 ←0　でマスクできる
3203	カナ出力	RS-232C ポートに SI/SO プロトコルでカナを出力する。
3226	RS-232C 一文字出力	COM1に1文字出力する。Acc←文字コード　&　CALL
3242	キースキャン	キースキャンを行なう。

アドレス	項目名	機能
3242	キースキャン	CY＝1　キー押されていない。 ＝0，Z＝1　キー押したまま　EEFEに ＝0，Z＝0　新しいキーを押した　データ
3583	キー入力	キーボード（正確にはキー入力用キュー）から1文字入力して Acc に入れる。入力待ちあり。
35A8	COPY 処理	COPY キーの処理
35C2	ブレークチェック	STOP, ^C のときキャリーをセットして戻る。
35CE	キー入力	キーバッファから1文字読む。 Z ：バッファエンプティ NZ：Acc ←入力データ
35D9	キーバッファ イニシャライズ	キー入力用キューをイニシャライズ（クリア）する。 このルーチンを用いる場合、DI 状態で行なって下さい。
369A	DISKI／0	物理的 DISKI／0 ルーチン entry： EC85　←ドライブ#（0～） EF50　←ドライブタイプ 8インチ DMA タイプ………………0 5インチ DMA タイプ………………1 5インチ片面インテリジェント……2 5インチ両面インテリジェント……3（内蔵ドライブ） ECB4　←0（エラーカウント） HL　←データアドレス BC　←トラック#，セクタ# フラグ　←I／0モード CY＝1…………ライト CY＝0，Z……チェック CY＝0，NZ……リード exit： CY＝1：エラー ＝0：正常終了，BCHLA レジスタは保存
36E2	サブシステム イニシャライズ	サブシステムをイニシャライズする。 exit：Acc ←ドライブ数
37C9	コマンド送出	サブシステムにコマンドバイトを送る。(Acc←コマンドしてCALL)
37D2	データ送出	サブシステムにデータバイトを送る。(Acc←データしてCALL)
3487	データ受信	サブシステムからデータを受けとる。 Acc にデータが入る。CY＝1のときはエラー
3A88	HEAD RESTORE	DMA タイプドライブのヘッドリストア Creg←××××× HD US1 US0 して CALL
3AAD	SEEK TRACK	DMA タイプドライブのトラックシーク Creg←××××× HD US1 US0 Dreg←トラック#　して CALL
3AF7	STATUS CHECK	DMA タイプドライブのステータス読み込み。
3C71	インタラプト待ち	FDC からのインタラプトを待つ。
3C7F	リードFDC	FDC からデータを読む。
3C94	ライトFDC	FDC へデータを書き込む。
3CAC 3C39	DMA8インタラプト DMA5インタラプト	8インチ DMA タイプ　インタラプトルーチン 5インチ DMA タイプ　インタラプトルーチン
3DCB	ドライブタイプゲット	entry : Acc ←ドライブ#－1 exit　: Acc, EF5D＝ドライブタイプ
3E0D	CRT 出力	CRT への1文字出力（Acc＝ASCII コード）
3E9B	BELL	BEEP 出力

アドレス	項　目　名	機　　　　能
3 E B 4	BEEP	BEEP エントリ
3 E C 0	ビープ ON/OFF	Acc＝０でBEEP０、０以外でBEEP１
3 E D 4	プリンタ出力ハンドラ	プリンタへ１バイト送る。(Acc＝ASCII コード)
3 F 2 7	プリンタESC＋出力	プリンタに ESC＋Acc を送る。
3 F 3 1	CSRL IN	CSRL IN→FAC
3 F 3 F	PEN 割り込み チェック	PEN 割り込みを起こすかどうかのチェック
3 F 6 2	PEN リード	ライトペン位置読み込み　H, L ←X, Y
3 F 7 9	ファンクションキー 表示	ファンクションキー（f･１～f･５）の表示をする。 （E6B8）＝０のとき表示しない。 ≠０のとき表示する。
3 F 7 A	ファンクションキー 表示	ファンクションキーの表示。(E6B8)＝０のときは表示しない。 Acc＝０のときf･１～f･５ Acc＝５のときf･６～f･10を表示する。
3 E B 4	BEEP	BEEPエントリ
3 E C 0	ビープON/OFF	Acc＝０で BEEP０、０以外でBEEP１
3 E D 4	プリンタ出力ハンドラ	プリンタへ１バイト送る。(Acc＝ASCII コード)
3 F 2 7	プリンタESC＋出力	プリンタに ESC＋Acc を送る。
3 F 3 1	CSRL IN	CSRL IN→FAC
3 F 3 F	PEN 割り込み チェック	PEN 割り込みを起こすかどうかのチェック
3 F 6 2	PEN リード	ライトペン位置読み込み　H, L ←X, Y
3 F 7 9	ファンクションキー 表示	ファンクションキー（f･１～f･５）の表示をする。 （E6B8）＝０のとき表示しない。 ≠０のとき表示する。
3 F 7 A	ファンクションキー 表示	ファンクションキーの表示。(E6B8）＝０のときは表示しない。 Acc＝０のときf･１～f･５ Acc＝５のときf･６～f･10を表示する。
4 0 1 5	ONインタラプトベクト トルアドレスのゲット	entry : Acc ←ONインタラプト＃ exit　: HL ← インタラプトベクトルアドレス
4 0 2 1	最下行クリア	テキスト画面の最下行をクリアする。
4 0 4 7	タイマリード	タイムバッファに時間を読み込む。 exit： （F00D）　秒（BCD） （F00E）　分（BCD） （F00F）　時（BCD） （F010）　日（BCD） （F011）　月（バイナリー） （F012）　年（BCD）
4 1 2 1	INPUT	(LINE) INPUT WAITのWAITの処理をする。
4 1 4 3	CLOCKインタラプト	CLOCK インタラプトルーチン
4 2 4 2 4 2 5 0	クロック割込み禁止 クロック割り込み許可	クロック割り込みを禁止する。 クロック割り込みを許可する。
4 2 5 7	VRAMアドレス	VRAM上の各行の先頭アドレス－VRAMTOPの値が格納されている。
4 2 8 B **4 2 9 0**	カーソル　OFF カーソル　ON	カーソルを表示しない。 カーソルを表示する。
4 2 9 D	VRAMアドレス	カーソル位置(H, L)→カーソルアドレス(HL)の計算 カーソル位置は１～。HL以外のレジスタは変わりません。
4 2 B 4	ヌルライン イニシャライズ	ヌルライン（通常FF80～FFF7）をイニシャライズする。
4 2 D 4	UPスクロール	テキスト画面をUPスクロールする。

アドレス	項 目 名	機 能
42FE	DOWNスクロール	テキスト画面をDOWNスクロールする。
431E	ラインアドレス	entry : L←テキスト画面 行#（1～） exit : DE=行の先頭のアドレス
433F	VRT 同期	Vertical retrace になるまで待つ。
434C	VRAM1 文字PUT	B ←キャラクタコード HL←VRAM上アドレス &CALL アトリビュートは COLOR 文で設定したもの。
4350	VRAM1 文字PUT	B ←キャラクタコード C ←アトリビュートコード HL←VRAM上アドレス &CALL
4351	アトリビュートセット	C ←アトリビュートコード HL←VRAM上アドレス &CALL
4452	VRAM1 文字GET	entry : HL ←VRAM上アドレス exit : Acc, B=キャラクタコード C=アトリビュートコード
445B	アトリビュート作成	entry : Acc←ファンクションコード（カラーコード） exit : Acc=アトリビュートコード
4472	スクリーン1文字GET	entry : H, L ←X, Y（1～） exit : Acc, B=キャラクタコード C=アトリビュートコード 他のレジスタは変わりません。
447D	スクリーン1文字PUT	スクリーンに1文字出力します。 entry : (EF87) =X座標（1～） (EF86) =Y座標（1～） Acc =キャラクタコード
449F	POS (X)	POS (X)→Acc
44A4	アドレス変換	実アドレス(HL)→ウィンドウ内アドレス(HL)
44B3	アドレス変換	実アドレス(BC)→ウィンドウ内アドレス(BC)
44D5	アドレス変換	ウィンドウ内アドレス(HL)→実アドレス(HL)
4551	ROM3 CALL	ROM3 の処理を CALL する。
458F	IPL ロード	IPL をロードし，ジャンプする。
45DD	PUT キュー	キューにデータを入れる。 Acc←キュー#, E←データ&CALL
45F8	GET キュー	キューからデータを読み出す。 entry : Acc←キュー#（0 or 1） exit : Z : キューエンプティ NZ : Acc=データ
461B	POP キュー	PUT キューを行わなかったことにする。 entry : Acc←キュー#（0 or 1） exit : Z : キューエンプティ NZ : Acc=PUT したデータ
463D	キューの イニシャライズ	キューのイニシャライズ（クリア）をする。 entry : Acc←キュー# B ←キュー長（2^n-1） DE ←キューアドレス
464E	BACK キュー	データをキューの TOP に入れる。 entry : Acc←キュー# E ←データ
4655	キュー・データ長	キューに入っているデータの数を HL に入れて戻る
4667	FRE キュー	キューの空いているバイト数を HL, Acc に入れて戻る
4680	キュー·テーブル·アドレス	entry : Acc←キュー#

アドレス	項　目　名	機　　　　能
4680	キュー・テーブル・アドレス	exit　: HL ←キュー・テーブル・アドレス
468C	ファイル ディスクリプタ処理	ファイル・ディスクリプタの処理をする。 entry : HL←ファイル・ディスクリプタの前 RST 10H をした後CALLする exit　: D　　＝デバイス＃ EC8F～＝ファイル名
46F8	VARPTR (＃n)	＃Acc の VARPTR を HL に入れる。
4798	OPEN	OPEN エントリ
481D	ファイル CLOSE	＃Acc のファイルを CLOSE する。
4852	RUN" "	RUN<ファイル>エントリ
4854	LOAD	LOAD エントリ
4855	MERGE	MERGE エントリ
49AA	RSET	RSET エントリ
49AB	LSET	LSET エントリ
4A5C	FIELD	FIELD エントリ
4AA1 4AA4 4AA7	MKI$ MKS$ MKD$	MK○$ (FAC) →FAC
4ABA 4ABD 4AC0	CVI CVS CVD	CV○ (FAC) →FAC
4B04	CLOSE	CLOSE エントリ
4B0C	CLOSE ALL	すべてのファイルを CLOSE する。
4B41 4B42	PUT ＃n GET ＃n	PUT ＃n処理ルーチン GET ＃n処理ルーチン
4B54	ファイル出力	カレントファイルにデータを出力する。
4B7B	ファイル入力	カレントファイルからのデータのシーケンシャル入力。
4BAC	INPUT$	INPUT$ エントリ
4C2F	LOC	LOC (FAC) →FAC
4C40	LOF	LOF (FAC) →FAC
4C51	EOF	EOF (FAC) →FAC
4C62	FPOS	FPOS (FAC) →FAC
4CC1	LINE INPUT＃	LINE INPUT＃ 処理ルーチン
4DA0 4DA3 4DA6 4DA9 4DAC 4DAF 4DB2 4DB5 4DB8 4DBB 4DBE 4DC1		Bad file name File already open Direct statement in file File not found File not open FIELD overflow Bad file number Internal error Input past end Sequential after PUT Sequential I/O only Feature not available
4E53	デバイス名←→ デバイス＃　テーブル	デバイス名とそのデバイス＃が入っている。
4E7C	処理ルーチン テーブルアドレス	デバイスごとの I/O 処理ルーチンテーブルの先頭 アドレスが入っている。
4E8C	シーケンシャル デバイス各種処理	DISK以外のデバイスに対する各処理ルーチンに分岐する。 entry : Acc ←処理＃×2

アドレス	項　目　名	機　　　　能
4E8C	シーケンシャル デバイス各種処理	HL ←VARPTR(＃n) D ←デバイス＃ その他処理により他のレジスタ, フラグ, スタックにパラメータが必要 処理＃　処理 0　OPEN 1　CLOSE 2　PUT/GET 3　OUTPUT 4　INPUT 5　LOC 6　LOF 7　EOF 8　FPOS 9　BACK READ DATA 10　WIDTH
4EC1	OUT of memory チェック	C ←必要なスタックレベル & CALL
4F01	NEW	NEW を行なう。
4F21	CLEAR	変数の CLEAR を行なう。
4FE5 4FF5 4FFB 5012	ONインタラプトON ONインタラプトOFF ONインタラプトSTOP ONインタラプト リクエスト	ONインタラプトをONにする。 ONインタラプトをOFFにする。 ONインタラプトをSTOPにする。 ONインタラプトをリクエストする。 entry : HL ←割り込みベクトルアドレス. (cf. 4015)
5045	ALL OFF	すべてのONインタラプトをOFFにする。
5059	ポーリング	ONインタラプトのポーリングをして GOSUB する。
50A5	RESTORE	RESTORE エントリ
50CA	STOP	STOP エントリ
50E5	END	END エントリ
5140	CONT	CONT エントリ
5158 5159	TRON TROFF	TRONエントリ　TRON フラグ=EC3B TROFF エントリ
515E	SWAP	SWAP エントリ
519C	ERASE	ERASE エントリ
522E	CLEAR	CLEAR エントリ
52BD	NEXT	NEXT エントリ
537D	ラベルサーチ	ラベルテーブルからラベルのサーチをする。
53AF	ラベル長カウント	ラベルの長さを数える。
53F6	ラベル登録	プログラム中のラベルをすべて登録する。
5441	ラベルリファレンス	ラベル名→ラベルのついている行のアドレス entry : HL ←テキストポインタ (＊の所) exit : HL ←テキストポインタ (ラベルの次) DE ←ラベルのついている行のアドレス
5480	ラベルスキップ	
5494	ストリング比較	
54C1 54C6 54CB	OCT$ HEX$ STR$	OCT$ (FAC) →FAC HEX$ (FAC) →FAC STR$ (FAC) →FAC
54D8	ストリングコピー	文字列領域にコピーの文字列をつくる。 entry : HL ←ソースストリングのディスクリプタのアドレス exit : DE ←コピーストリングのディスクリプタのアドレス

アドレス	項　目　名	機　　　　能
54EE	ストリング スペース確保	Acc 個のストリングスペースを確保する。 HL にそのディスクリプタのアドレスが入る。
5500	ストリング登録	(HL＋1)～00 or Breg or Dreg をストリングとしてストリングスタックに PUSH し、FAC に入れる。
5530	ストリング登録	DE が指すディスクリプタの文字列をストリングスタックに PUSH し、FAC に入れる。
5550	文字列出力	(HL) ～00の文字列を出力する。
5572	ストリング スペース確保	Acc 個のストリングスペースを確保する。 DE にそのアドレスが入る。ディスクリプタは作られない。
559A	ガベージコレクション	ガベージコレクションを行なう。
56BA	ストリングコピー	HL ←ソースストリングのディスクリプタアドレス DE ←ディスティネーションスペースのアドレス
56C9	チェック & POP	FAC が文字型であることを確認してからストリング POP する。
56CC	ストリング POP	FAC のストリングをストリングスタックから POP する。
56F8 56FC	LEN	LEN (FAC) →FAC →Acc
5704 5708	ASC	ASC (FAC) →FAC →Acc
5714	CHR$	CHR$ (FAC) →FAC
5722	STRING$	STRING$ エントリ
5741	SPACE$	SPACE$ (FAC) →FAC
575A 578A 5793	LEFT$ RIGHT$ MID$	LEFT$エントリ RIGHT$エントリ MID$関数エントリ
57B4	VAL	VAL (FAC) →FAC
57D7 **5816**	INSTR INSTR	INSTR エントリ INSTR (P, A$, B$) →Acc entry : Acc ←P DE ←VARPTR (B$) HL ←VARPTR (A$) exit : Acc ←INSTR
585A	MID$文	MID$文エントリ
58E4	FRE	FRE (FAC) →FAC
5935	プリンタ出力	プリンタへ1文字出力する。
5989	プリンタ改行	LPOS≠0のときプリンタを改行する。
59A4	コンソール出力	コンソールへ1文字出力する。
5A08	1文字入力	入力デバイスから1文字入力する。
5A58	改行	POS (X) ≠0のとき改行する。
5A86	イベントキーチェック	イベントキー (^O, ^S, ^C) が押されていたら，それに応じた動作を行なう。
5AA3 **5AA4**	INKEY$	INKEY$エントリ INKEY$→FAC
5AC5	DIM	DIM エントリ
5ACA	変数アドレス GET	entry : HL←テキストポインタ（変数名の最初） exit : DE←VARPTR
5DD5	カーソル移動コード 処理アドレステーブル	LF, HOME, CLR, CR, →, ←, ↑, ↓, の順 カーソル移動処理を行なうルーチンのアドレステーブル
5DE5 5DF7 5E03 5E23	カーソル進める カーソル復帰 カーソル改行 カーソルDOWN	→ CR LF ↓

アドレス	項　目　名	機　　　　　能
5E33	カーソルUP	↑
5E49 5E54 5F0E	ホームポジション カーソル後退 画面クリア	HOME ← CLR
5F76 5F86	行継続コード読み込み 行継続コード書き込み	L行(1〜)が次の行とつながっているかどうかを示すコードを読み込む。 cf. EF9A 〜 行継続コードを書き込む。
5F92 5FC8	1行入力(1) 1行入力(2)	カーソルのある行全部を読み込む。 カーソル位置から入力した所までを読み込む。 exit : E9B9〜　入力文字列 CY＝1：STOP, ^C ＝0：リターンキー
657B	EDIT	EDIT エントリ
6642	PRINT USING	PRINT USING 処理
67EF	カセットセーブ	カセットへのプログラムセーブを行なう。 F009＝EBH　1200ボー ＝FAH　600 ボー
680F	カセットロード	カセットからのロード，ベリファイ Acc ＝ 0　　ロード ＝FF　　ベリファイ
69B2	フックイニシャライズ	フックをイニシャライズする。
69D1	インタラプトベクトル イニシャライズ	インタラプトベクトル（F300〜F31F）をイニシャライズする
69E1	CRTイニシャライズ	CRT をイニシャライズする。
69FE	処理ルーチン アドレステーブル	中間言語81H〜D9H に対する処理ルーチンのアドレステーブル
6AB0	FFシリーズ前半の 処理ルーチン アドレステーブル	中間言語FF81〜FFA9の関数に対する処理ルーチンのアドレステーブル
6B02	FFシリーズ後半の 処理ルーチン アドレステーブル	中間言語 FFD0〜FFE4H に対する関数・文としての処理アドレステーブル 4バイトで1組 前の2バイト……関数としての処理ルーチンアドレス 後の2バイト……文としての処理ルーチンアドレス
6B56	予約語のインデックス	予約語テーブルの最初の1文字による INDEX
6B8A	予約語テーブル	予約語とその中間言語のリストが入っている。
6E81	演算子テーブル	1文字の予約語とその中間言語のリスト。
6E96	ROM3処理呼び出し	ROM3処理を呼び出すために使われる。4バイトで1組。 CALL　4551H DB　　<処理＃> これを実行することにより<処理＃>に対応する処理が行なわれる。 ROM3ルーチン一覧表参照
6F12	エラーメッセージ	ROM-BASICのエラーメッセージテーブル
6F6B	テキスト画面 WIDTH	テキスト画面のイニシャライズ（モード変更） entry : B ←桁数 C ←行数 (E6B2)　スクロール開始行 (E6B3)　スクロール終了行 (E6B4)　ヌルアトリビュート (E6B9)　00：白黒モード、FF：カラーモード

アドレス	項目名	機能
6FD1	CRTC セット	CRTC, DMAC をセットする。 entry : CY　　＝1（40桁）、0（80桁） (E6B9)　＝FF（カラー）、00（白黒） (E6C4, 5) ＝VRAM TOPアドレス (F3C8) HL　　＝705B（20行）、7066（25行）
7071	CONSOLE	CONSOLE エントリ
714E	LOCATE	LOCATE エントリ
7198	GET	GET エントリ
71A6	PUT	PUT エントリ
71B5 71C6	CLS CLS	CLS エントリ B ←＜機能＞ & CALL
71D9	タイマセット	タイムバッファ (F00D～F011) のデータで時間をタイマにセットする（cf. 4047)
721C	DATE$	DATE$ 文エントリ
7279	TIME$	TIME$ 文エントリ
72AB	HELP	HELP 文エントリ
72B0	STOP	STOP ON /OFF /STOP処理
72C8	PEN	PEN 文エントリ
72CD	I/O イニシャライズ	N_{88}-BASIC 用I/O をイニシャライズする
7348	TERMサブルーチン	
7367	TERM	TERM エントリ
7570	COM1ポーリング	COM1をポーリングする。HL←受信した文字数
779C	モード分岐	イニシャライズ時にターミナルモード，N-BASICモードへ分岐するルーチン。
77DD	NEW	NEW エントリ
77E5	NEW ON	DE←（NEW ON値）してここへ JMP する
77F7	システム イニシャライズ	リセットから飛んでくる。
7984		文字列 'How many files(0-15)', 0
79B2	クレジット	文字列 'Bytes free', 0 文字列 'NEC N-88 BASIC ……', 0
79FC	I/O処理ルーチン アドレス	DISK以外の I/O 処理ルーチンのアドレステーブル 各デバイスにつき22バイト，11個の処理で LPT1，COM1，COM2·3，CAS1, CAS2, SCRN, KYBD の順 (cf. 4E7C, 4E8C)
7A96	OPEN	FCB を OPEN 状態にする。
7AC0	GET/PUT の大きさ	DISK以外のデバイスに対するGET/PUTの大きさの処理ルーチン
7B25	LPT FPOS	LPT FPOS　処理
7B39	LPT OPEN	LPT OPEN　処理
7B59	LPT CLOSE	LPT CLOSE　処理
7B76	LPT PUT	LPT PUT　処理
7BA5	LPT OUTPUT	LPT OUTPUT　処理
7BBE	LPT WIDTH	LPT WIDTH　処理
7BC2	COM1 OPEN	COM1 OPEN　処理
7C3B	COM1 INPUT	COM1 INPUT　処理
7C43	COM1 OUTPUT	COM1 OUTPUT　処理
7C57	COM1 CLOSE	COM1 CLOSE　処理
7C6E	COM1 LOC	COM1 LOC　処理
7C76	COM1 LOF	COM1 LOF　処理
7C7E	COM1 EOF	COM1 EOF　処理
7C8B	COM1 BACK	COM1 BACK READ DATA

アドレス	項　目　名	機　　　　能
7C96 7C9B	COM1 WIDTH COM1 1文字INPUT	COM1 WIDTH　処理 Acc ← 1 & CALL Acc データ
7CCD 7D71	 COM	'Line buffer overflow' エラー出力 COM エントリ
7DC3 7E3A 7E41 7E7B 7E82 7E8A	カセットOPEN カセットCLOSE カセットGET/PUT カセットBACK カセットOUTPUT カセットINPUT	カセットOPEN　処理 カセットCLOSE　処理 カセットGET/PUT　処理 カセットBACK READ DATA カセットOUTPUT　処理 カセットINPUT　処理
7EBA	ボーレイト	カセットボーレイト設定(F009=FA : 1200ボー，FB : 600ボー)
7ED0	カセットREAD ON	カセットリードをスタートする (F009=FA : 1200ボー，FB : 600ボー)
7F15	カセットREAD OFF	カセットリード中止
7F1A	カセットWRITE OFF	カセットライト中止
7F30 7F35	MOTOR MOTOR	MOTOR エントリ カセット用MOTOR の ON/OFF Acc ≠ 0 : MOTOR ON = 0 : MOTOR OFF
7F4D	カセットWRITE ON	カセットライト用イニシャライズ (F009=FA : 1200ボー，FB : 600ボー)
7F87	カセットリード	カセットから1文字入力する。データ→Acc
7FD0	カセットライト	カセットへ1文字入力する。Acc←データ & CALL

B．4 th ROM 1　部分（アドレス6000～7FFF）

アドレス	項　目　名	機　　　能
600D	ROM3ジャンプテーブル（2バイト×31組）	メインROMでのROM3コール CALL　4551 DB　＜処理番号＞ での、処理番号に対するジャンプテーブル（別表）
604B	CIRCLE文サブルーチンⅠ	スクリーンモードに合わせて縦、横の比を求める。
605D	GET @文、PUT @文サブルーチン	GET @文、PUT @文のサブルーチンで、各種ワークの設定、および実際のパターンの読み出し、書き込みを行なう。
6260	スクリーンモードチェックⅠ	スクリーンモードによって、Accをセットする。 カラーモード……Acc＝3 B/Wモード………Acc＝1
626A	PAINT文サブルーチンⅠ	PAINT文のサブルーチンの集まり。
6392	境界色セット	PRINT文の境界色をセットする。 entry：Acc←パレット番号
63A6	PAINT文サブルーチンⅡ	PAINT文のサブルーチンの集まり。
64C0	VIEW PORT内チェック	オリジナル・スクリーン座標（X,Y）（BC,DEレジスタ）がVIEW PORT内にあるかどうかをチェックする。 内……CY＝1　外……CY＝0
6509	CIRCLE文サブルーチンⅡ	FAC＝FAC * FRXの計算（FACには、円の半径が入っている）
6518	グラフィックアドレスの計算	オリジナルスクリーン座標からグラフィック画面の物理アドレス、マスクパターンを求める。 entry：BC←X座標 DE←Y座標 exit：（F033,34）＝アドレス （F035）＝マスクパターン
657C	LINE文サブルーチンⅠ	LINE文で、点を上下左右に移動させたときの、画面アドレスを求める。（一部、PAINT文でも使用）
65F2	汎用サブルーチンⅠ	
661B	POINT(Sx,Sy)関数サブルーチン	6518のルーチンでセットした場所の点の状態（カラーコード）をAccに入れてもどる。
6647	LINE BF文サブルーチン	LINE BF文で指定した領域を塗りつぶすサブルーチン。
66A0	カラーマスクセット	パレット番号によって（F036～38）に00またはFFを書き込む。 entry：Acc←パレット番号
66D7	グラフィックデータ書き込み	上のルーチンでセットしたカラーで、6518のルーチンでセットした位置に点を書く。
6700	グラフィックスクリーン初期化	リセットやホットスタート時に呼び出される。グラフィックスクリーンをクリアし、各種ワークエリア等を初期化する。 entry：（F087）←スクリーンモード（0,1,2） （F088）←画面スイッチ（0,1,2,3）
6878	COLOR文	COLOR文の処理を行なう。 COLOR…6882 COLOR＝…68EC COLOR @…6927
698F	SCREEN文	SCREEN文の処理を行なう。

アドレス	項　目　名	機　　　　　能
6A94	CLS2文	CLS2文の処理を行なう。
6AC6	VIEW文	VIEW文の処理を行なう。
6C2B	WINDOW文 サブルーチン	WINDOW文の座標パラメータを得る。
6C55	WINDOW文	WINDOW文の処理を行なう。
6CA8	VIEW関数	VIEW関数の処理を行なう。
6CD5	WINDOW関数	WINDOW関数の処理を行なう。
6D0A	LPの初期化	LPの値を左上の座標にセットする。 （SCREEN文、VIEW文、WINDOW文の実行後）
6D29	MAP関数	MAP関数の処理を行なう。 W→S…6D8B S→W…6DAB
6DC0	POINT（　）関数	POINT文の処理を行なう。
6E25	POINT文	POINT文の処理を行なう。
6E2B	座標パラメータ （ワールド座標系）	テキストから座標パラメータ（ワールド座標系）を得る。（WINDOW文が実行されていなければ、スクリーン座標系となる）
6EF8	スクリーン座標 →ワールド座標	スクリーン座標からのワールド座標への変換を行なう。 entry：(F027,28),(F029,2A)スクリーン座標のX,Y（LP） exit　：(F0A4～A7)　ワールド座標のX（LP） (F0A8～AB)　ワールド座標のY（LP）
6F33	LINE文サブルーチンII	LINE文で、線がオリジナルスクリーン上に書けるかどうかを調べる。
7053	画面コピー	画面コピー（COPY文、COPYキー）の処理を行なう。
705A	COPY	画面コピーする entry：Acc←コピーモード(1～5)
723D	漢字データ読み出し	PUT @ KANJI文のサブルーチンで漢字データをワークエリアの上に読み出す。
7261	漢字フォント読み出し	漢字フォントをE9B9～に読み込む（p ??? 参照） entry：DE←漢字フォント exit　：E9B9～漢字フォント
72D0	ON ×× GOSUB サブルーチンI	Line #（DEレジスタ）をAcc番目のジャンプテーブルにセットする。
72EC	Read Light Pen	ライトペン入力信号が来たときの行、列座標を読み出す。
72ED	PEN関数	PEN関数の処理を行なう。
7324	ON ×× GOSUB サブルーチンII	ON ×× GOSUBの前処理を行なう。××によって、BCレジスタをセットして、メインROMへ戻る。
73DF	KEY文	KEY文の処理を行なう。 KEY LIST………………73DF KEY 定義……………………7443 KEY OFF…………………7484 KEY ON…………………748D KEY STOP………………749C KEY ON, OFF, STOP …74A1
74EE	DATE$関数	DATE$関数の処理を行なう。
752A	TIME$関数	TIME$関数の処理を行なう。
754F	ドライブ対応表作成	ドライブ番号→ドライブタイプの対応表を作成する。
75A1	両面モードセット	サブシステムを両面モードにする。
75DD	RENUM文	RENUM文の処理を行なう。
7674	PAINT文	PAINT文の処理を行なう。
76A4	PAINT	PAINTを行なう ［CALLの方法］（ただし、WINDOWやVIEWを使わないとき） A．タイルペイントを行なわないとき

アドレス	項　目　名	機　　　　　　能
76A4	PAINT	①ROM3をアクティブにする ②7939のルーチンでPAINT用のキューを初期化する ③66A0のルーチンで領域色をセットする ④6392のルーチンで境界色をセットする ⑤リターンアドレス（⑩の位置）をPUSHする ⑥X座標をスタックにPUSHする ⑦Y座標をスタックにPUSHする ⑧（F056）に0を入れる ⑨このルーチンにジャンプする ⑩メインROMに戻す B．タイルペイントを行なうとき ①Aの①②③④⑤⑥⑦をおこなう ②（F056）に1を入れる ③以下のワークエリアをセットする (F04F,50),(F052,53),(F057),(F054,55) ④このルーチンにジャンプする ⑤メインROMに戻す
77DE	CIRCLE文 サブルーチンⅢ	DE＝－DEを求める。
77E4	PAINT文 サブルーチンⅢ	PAINT文のサブルーチン。
7939	PAINTキュー初期化	PAINT用のキューの初期化を行なう。 必要があるときはガベージコレクションを行なうので注意
7971	PAINT文 サブルーチンⅣ	
79D6	CIRCLE文	CIRCLE文の処理を行なう。
7A69	CIRCLE処理	CIRCLEを行なう。 ［CALLの方法］（ただし、WINDOWやVIEWを使わないとき） ①ROM3をアクティブにする ②66A0のルーチンで色をセットする ③(F027,28),(F029,2A)にX座標、Y座標をセットする ④(F01A,1B)に半径をセットする ⑤以下のワークエリアをセットする (F072,73),(F074,75),(F076,77),(F07A),(F07B),(F07C,7D),(F084) ⑥リターンアドレス（⑨の位置）をPUSHする ⑦HLをPUSHする（ダミー） ⑧このルーチンにジャンプする ⑨メインROMに戻す
7C33	GET@，PUT@文	GET@，PUT@文の処理を行なう。 GET@……(F071)＝0 PUT@……(F071)＝1
7D98	座標パラメータ （スクリーン座標系）	テキストから座標パラメータ（スクリーン座標系）を得る。
7DFB	PRESET,PSET文	PRESET文の処理を行なう。…7DFB PSET文の処理を行なう。……7E00
7E19	POINT(Sx,Sy)関数	POINT(Sx,Sy)関数の処理を行なう。
7E34	パレット番号を得る	テキストからパラメータ（パレット番号）を得る。
7E55	汎用サブルーチンⅡ	HL＝(F01A,1B)－BC……7E55 HL＝(F01C,1D)－DE……7E67 DE←→(F01C,1D)……7E72 BC←→(F01A,1B)……7E7F

アドレス	項　目　名	機　　　　能
7E8B	LINE文	LINE文の処理を行なう。 [LINEの引き方] ①ROM3をアクティブにする ②66A0のルーチンでカラーをセットする ③(F025,26)にラインスタイルをセットする ④(F01A,1B),(F01C,1D)に終点のX座標、Y座標を入れる ⑤BC, DEに始点のX座標、Y座標を入れる ⑥処理に応じて以下のどれかをCALLする LINE BF ……7EB7 LINEのみ……7EF5 LINE B ………7F0E ⑦メインROMに戻す
7EBD	汎用サブルーチンⅢ	DE＝DE ¥2
7FC5	LINE文 サブルーチンⅢ	ラインスタイルを見て、点を打つ。
7FD5	汎用サブルーチンⅣ	テキストから1バイトのパラメータを得る。
7FED	専用高解度 ディスプレイチェック	専用高解度ディスプレイモードかどうかを調べる。

【ROM3】処理番号と処理開始アドレス

処理番号	アドレス	内　　容
0	7DFB	PRESET 文
1	7E00	PSET 文
2	6D29	MAP 関数
3	7C33	GET @, PUT @文
4	7053	COPY 文
5	7064	COPY キー
6	7E8B	LINE 文
7	6E25	POINT 文
8	6700	グラフィック画面初期化
9	6AC6	VIEW 文
10	7E19	POINT (Sx, Sy)関数
11	6DC0	POINT 関数
12	6878	COLOR 文
13	EEBC	ROLL 文
14	79D6	CIRCLE 文
15	6CD6	WINDOW 関数
16	6C55	WINDOW 文
17	7674	PAINT 文
18	698F	SCREEN 文
19	6CA8	VIEW 関数
20	6A94	CLS2
21	6D0A	LP の初期化
22	72D0	ON ×× GOSUB サブルーチン I
23	72ED	PEN 関数
24	7324	ON ×× GOSUB サブルーチン II
25	742E	KEY 文
26	752A	TIME$ 関数
27	74EE	DATE$ 関数
28	75A1	サブシステムモードセット
29	754F	ドライブ対応表作成
30	75DD	RENUM 文

C．ディスクコード

アドレス	項　目　名	機　　　　能
87D6 87E6	ROLL UP 後処理 ROLL DOWN 後処理	キーバッファをクリアして、EDIT へ戻る。 キーバッファをクリアして、EDIT へ戻る。
87F4	CHAIN 文サブルーチン	ラベルスキップ
8803	ROLL 文	ROLL 文の処理を行なう。
88E9	SEARCH 関数	SEARCH 関数の処理を行なう。
89B3	ATN 関数（単精度）	ATN(FAC)→ FAC
89D7	データ	ATN 関数用のデータ
89FC	ディスク情報セット	ディスク情報をワークエリア(ECA5～)にセットする。
8A18	データ	ディスク情報データ
8A3C	エラーメッセージ	フルセンテンスエラーメッセージ
8DE5	テキスト入力時の処理	MAIN ループでのテキスト入力時に、CHAIN 文であれば、ストリングエリアをクリアしないようにするためのルーチン。
8E4B	CHAIN クリア	エラーがおこったとき、CHAIN 文フラグをクリアする。
8E55	デバイス　デフォルトセット	ファイルディスクリプタのデバイス番号のデフォルト値を0にする。
8E58	イニシャライズ	ドライブテーブルを初期化して、ID セクタを実行する。
8E9A	GET デバイス＃	ファイルディスクリプタのデバイス＃を Acc へ入れる。
8EC3	DSKF 関数	DSKF 関数の処理を行なう。
8F41	式の評価	式の評価ルーチンを倍精度関数用に拡張
8F7C	倍精度分岐	倍精度関数ルーチンへのふり分けを行なう。
8F8E	倍精度関数ジャンプテーブル	SQR, RND, SIN, LOG, EXP, COS, TAN, ATN
8FA4	DISK 関数評価	DSKI$, INPUT$, ATTR$ の分岐 また、USR 関数実行時に、プロテクトのチェックを行なう。
8FBD	ファイル関数評価	EOF, FROS, LOC, LOF の分岐
8FD0	フルセンテンスエラーメッセージセット	フルセンテンスエラーメッセージのポインタをセットする。
9001 900C 9018 9032 903C	KYBD ：OPEN KYBD ：INPUT＃ KYBD ：GET＃ KYBD ：READ BACK KYBD ：LOC	OPEN “KYBD ：” 処理 KYBD の INPUT＃処理 KYBD の GET＃処理 KYBD の READ BACK 処理 KYBD の LOC 関数処理
9043 9051 905F	SCRN ：OPEN SCRN ：OUTPUT SCRN ：PUT＃	OPEN “SCRN ：” 処理 SCRN の OUTPUT＃処理 SCRN の PUT＃処理
90C4	WEND サーチ	WEND をサーチする。
90C9	ワークエリア	CHAIN 文で使用する。OPTION BASE のフラグと値
90CC 9118 91A5	ROLL UP ROLL DOWN EDIT 行番号 GET	EDIT モードでの ROLL UP キーの処理 EDIT モードでの ROLL DOWN キーの処理 EDIT モードのための行番号を画面から読みとる。
9247	倍精度関数サブルーチン	倍精設関数計算のための汎用サブルーチン集
9326	数値データ	倍精度関数計算のための数値データ
948B 94E7 94F0 956F 9615 9650	ATN（倍精度） COS（倍精度） EXP（倍精度） LOG（倍精度） SIN（倍精度） SQR（倍精度）	ATN (FAC)→ FAC COS (FAC)→ FAC EXP (FAC)→ FAC LOG (FAC)→ FAC SIN (FAC)→ FAC SQR (FAC)→ FAC

アドレス	項　目　名	機　　　　　　能
9691	TAN（倍精度）	TAN（FAC）→FAC
9717	ファイル削除	FATからファイルを削除する。
973B	MOUNT メインルーチン	ディスクをマウントする。
979F	文字列	DB 'Copies of allocation bad on drive',0
97FD	REMOVE サブルーチン	REMOVEルーチンのサブルーチン、FATの更新を行なう。
97EA	REMOVE	リムーブ処理を行なう。（OPEN,CLOSE,KILL） entry ：Acc←デバイス#
9821	MOUNT	デバイスがディスクだったらマウント処理を行なう。
982C	ファイル#チェック	PUT#,GET#で、ファイル#が現在のファイルのレコード数より大きいかどうかを調べる。
9869	LOC関数サブルーチン	クラスタ数を数える。
9886	FAT TOP セット	DEレジスタにFATの先頭番地をセットする。
98DA	クラスタ→セクタ	クラスタ数からセクタ数を求める。 entry ：C＝クラスタ数 exit ：DE＝セクタ数
98EA	ディレクトリサーチ	ディレクトリからファイル名を探す。（なければZ＝0）
9953	ディスク情報セット	ディスク情報、ドライブポインタをワークエリアにセットする。 entry ：Acc＝ドライブ#
9997	IDセクタ計算	BCレジスタにIDトラック，セクタをセットする。
99A1	ディレクトリトラック	Bレジスタにディレクトリトラックをセットする。
99B6	NAME文	NAME文の処理を行なう。
9A01	ディスクファイル OPEN	ディスクファイルをオープンする。
9B21	ディスクファイル CLOSE	ディスクファイルをクローズする。
9BAD	KILL文	KILL文の処理を行なう。
9BC5	ディスク SAVE	ディスクへのセーブルーチン
9C26	BSAVE サブルーチン	
9C35	BLOAD サブルーチン	
9C51	ディスク LOAD	ディスクからのロードルーチン
9CE7	OPEN チェック	ファイルがすでにオープンされているか調べる。
9DAD	SET文	SET文エントリ
9E1D	ATTR$関数	ATTR$関数エントリ
9E5E	SET文、ATTR$	SET文、ATTR$関数のパラメータを処理する。
9F2A	LFILES文	LFILES文エントリ
9F2F	FILES文	FILES文エントリ
9FFD	PUT#/GET#	ランダムファイルの読み書きを行なう。（ディスク）
A104	INPUT$	INPUT$文の処理（ディスク）
A185	DSKI$関数	DSKI$関数の処理を行なう。
A1C4	DSKO$文	DSKO$文の処理を行なう。
A1F3	DSKI$, DSKO$ サブルーチン	DSKI$, DSKO$のパラメータをレジスタにセットする。
A2F7	FAT READ	FATをメモリ上に読み出す。
A327	ディレクトリ READ	ディレクトリをメモリ上に読み出す。
A340	ディレクトリ WRITE	ディレクトリを書き込む。
A4E0	カレントセクタREAD	現在のファイルの1セクタを読み出す。
A4FF	トラック・セクタ計算	カレントクラスタ・セクタからトラック・セクタを計算する。
A53B	DISK READ/WRITE	ディスクのREAD/WRITEルーチン
A61A	DSKF関数	ディスクのフリークラスタを求める。
A640	LOC関数	
A669	LOF関数	

アドレス	項　目　名	機　　　　　能
A685	EOF 関数	
A6C9	FPOS 関数	
A6F1	ID セクタ READ	ID セクタをキーバッファに読み込む。
A710	エラー69	Bad allocation table
A713	エラー70	Bad drive number
A716	エラー71	Bad track sector
A719	エラー67	Disk already mounted
A71C	エラー63	Disk not mounted
A71F	エラー72	Deleted record
A722	エラー65	File already exists
A725	エラー68	Disk full
A728	エラー61	File write protected
A72B	エラー64	Disk I/O error
A72E	エラー62	Disk offline
A731	エラー73	Rename across disks
A73D	BSAVE 文	BSAVE 文の処理を行なう。
A7A8	BLOAD 文	BLOAD 文の処理を行なう。
A810	BLOAD, BSAVE サブルーチン	
A875	WHILE 文	WHILE 文の処理を行なう。
A89F	WEND 文	WEND 文の処理を行なう。
A916	CALL 文	CALL 文の処理を行なう。
A98C	CHAIN 文	CHAIN 文の処理を行なう。
AC10	CHAIN 文ジャンプ	CHAIN 文で指定された行番号へとぶ。
AC72	COMMON 文	COMMON 文の処理を行なう。
AC75	WRITE 文	WRITE 文の処理を行なう。
ACC1	SAVE 暗号化	P オプションセーブのときにプログラムを暗号化する。
AD05	LOAD 平文化	P オプションセーブをしたプログラムのロード後もとに戻す。
AD48	プロテクトチェック	プロテクト(P オプションセーブしたプログラムがロードされている)の状態であれば、以下のコマンドをエラーとしてしまう。 LIST, SAVE, MON, USR, CALL, PEEK, POKE
AF00	イニシャライズルーチン	最初にイニシャライズを行なったあとは不要になり、他の目的に使用される。

D．拡張命令パッケージ

アドレス	項　目　名	内　　　　　容
DC00	イニシャライズ	
DC13	CMD エントリ	
DC9C	サブコマンド解析	サブコマンド名を解析して各コマンドに飛ぶ
DCD5	サブコマンドアドレス対応表(1)	CMD SING サブコマンドとその処理アドレスの対応表
DD06	サブコマンドアドレス対応表(2)	CMD TURTLE サブコマンドとその処理アドレスの対応表
DD68	CMD TURTLE コマンド	CMD TURTLE の各サブコマンドの処理を行なう。 DD68　FD (Forward) DD75　BK (Back) DD8A　SX (Set X) DD93　SY (Set Y) DD9C　MV (Move) DDEC　RT (Right) DDF6　LT (Left) DE0A　HD (Head) DE48　CP (Complement) DE68　PD (Pen Down) DE69　PU (Pen Up) DE6E　PC (Pen Color) DE80　ST (Show Turtle) DE8E　HT (Hide Turtle)
DE9B	CMD SING コマンド	CMD SING の各サブコマンドの処理を行なう。 DE9B　…　T (Tempo) DEA7　…　L (Length) DEAE　…　O (Octave) DECC　…　R (Rest) DED4　…　B (play B) DED6　…　A (play A) DED8　…　G (play G) DEDA　…　F (play F) DEDB　…　E (play E) DEDD　…　D (play D) DEDF　…　C (play C)
DF5E	コピーライト	DB '(C) 1983 NEC ',0
DF6B	音階テーブル	CMD SING の各音階ごとの高さと長さの入ったテーブル
DFFE	X サブコマンド	X コマンドの処理をする
E00D	リピートサブコマンド	E00D　…　RP〔　（リピート開始） E048　…　〕　（リピート終了）
E06E	PUT TURTLE	タートルを書く
E120	ERASE TURTLE	タートルを消す
E17E	PSET XOR	XOR で PSET する
E194	LINE XOR	XOR で LINE を引く
E230	音階出力	実際に音を出すルーチン
E257	1バイト乗算	HL = A * C
E266	ROM3セレクト	ROM3をセレクトする
E26B	メインROM セレクト	メイン ROM をセレクトする
E272	次の位置の計算	距離と方向と LP から次の LP を計算する

アドレス	項　目　名	内　　　　容
E30D	値の読み込み	HLでさす数字列から値を計算してDEにいれる（整数）
E39E	CMD CLS エントリ	
E473	CMD TEXT ON/OFF エントリ	
E4B2	CMD CUT エントリ	

17-1-2 ワークエリア

A．拡張命令パッケージ

アドレス	項目名	機　能・用　途
E4D1	2	タートルの向き
E4D3	4	SIN（タートルの向き）
E4D7	4	COS（タートルの向き）
E4DB	4	タートルの大きさ（前後方向）
E4DF	4	タートルの大きさ（左右方向）
E4E3	1	ペンの色
E4E4	1	ペンの状態（0：PEN UP 1：PEN DOWN）
E4E5	1	補正するかどうか（1：する）
E4E6	1	タートルフラグ（0：表示しない　1：表示する）
E4E7	1	テンポ（Tサブコマンド）
E4E8	1	長さ　（Lサブコマンド）
E4E9	2	オクターブ（音階テーブルのそのオクターブの最初をさす）
E4EB	2	タートルの先頭のX座標
E4ED	2	タートルの先頭のY座標
E4EF	2	タートルの右足のX座標
E4F1	2	タートルの右足のY座標
E4F3	2	タートルの左足のX座標
E4F5	2	タートルの左足のY座標
E4F7	2	タートルの中心から右足までのX偏位
E4F9	2	タートルの中心から右足までのY偏位
E4FB	1	CMD SING/CMD TURTLEの切り換え（0：SING　1：TURTLE）
E4FC	1	負号フラグ
E4FD	24	リピートテーブル（3バイトで一組。リピートカウンタとリピートアドレス）
E515	1	リピートレベル
E516	2	リピートテーブルポインタ。最も深いレベルを指す。
E518	1	出力する音の高さ
E519	2	出力する音の長さ
E51B	1	スピーカを制御するビットマスク。休符のときは0
E51C	1	符点音符フラグ
E51D	3	CMD CLSでバックグラウンドカラーによる各プレーンに書き込む値(00 or FF)
E520	1	ポート31Hのバックアップ（CMD CLS用）
E521	1	ポート40Hのバックアップ（CMD CLS用）
E522	2	1プレーンクリアするサブルーチンのアドレス差

B．N88－BASIC 本体

アドレス	バイト数	機　能・用　途
E600	14	単精度除算用サブルーチン
E60E	1	RND 関数　発生カウンタ
E60F	1	RND 関数　ROM データ　カウンタ
E610	1	RND 関数　RAM データ　カウンタ
E611	32	RND 関数　RAM データ（単精度）4バイト×8組
E631	4	RND 関数の値
E635	20	USR 関数　アドレステーブル　2バイト×10組 （初期値　0B06H：Illegal function call）
E649	2	ERR の値（ERL = EB09）
E64A	1	未使用
E64B	1	LPOS の値
E64C	1	プリンタフラグ（出力先の指定．0：CRT,0以外：LPT）
E64D	1	LPRINT`，`改行桁数
E64E	1	WIDTH LPRINT の値（LPRINT`；`改行桁数）
E64F	1	PRINT　`，`改行桁数
E650	1	PRINT　`，`改行桁数
E651	1	未使用
E652	1	CTRL +0フラグ（画面出力をさせないためのフラグ）
E653	1	BLOAD/BSAVE 用フラグ（0：アドレス指定のみ　1：指定なし　FF：アドレス・ 長さあり）
E654	2	メモリ上限値、CLEAR 文の第2パラメータ
E656	2	実行中の行番号
E658	2	テキスト開始アドレス
E65A	2	OV,/0エラーメッセージアドレス
E65C		ターミナル ID
E669		モニタ用ワークエリア
E684		未使用
E69C	2	RST 10用 work（1E，10）
E69E	1	前にアクセスしたドライブタイプ(8インチ=0,5インチ SS =2,5インチ DS =3)
E69F	1	TERM 中を示すフラグ
E6A0	1	TERM リモートプロトコル用フラグ（FF：ESC〉とノーマル、00：ESC.,01：〈）
E6A1	1	TERM half キー入力1メッセージ最初の文字
E6A2	1	TERM LPT イネーブルフラグ（f・8）
E6A3	1	READ で0だったら Out of Data
E6A4	1	カセット（ロード中…FF, OPEN…1，1A を入力のとき…0）
E6A5	1	TERM LF/COPY フラグ（LF…1, COPY…2）
E6A6	1	ハイレゾモードフラグ（640＊200…0, 640＊400…1）
E6A7	1	カーソルフラグ
E6A8	1	カーソル ON/OFF コマンド
E6A9	1	カセット使用中フラグ（READ…1，WRITE…FF）
E6AA	1	未使用
E6AB	2	'　'の行番号
E6AD	1	INS モードフラグ
E6AE	2	ライトペン補正値
E6B0	1	テキスト画面ウィンドウ上限（実際のウィンドウ）
E6B1	1	テキスト画面ウィンドウ下限（実際のウィンドウ）
E6B2	1	テキスト画面ウィンドウ上限 CONSOLE A,B で→ A +1の値
E6B3	1	テキスト画面ウィンドウ下限 CONSOLE A,B で→ B +1の値
E6B4	1	ヌルアトリビュート

アドレス	バイト数	機　　　能・用　　　途
E6B5	1	ヌルキャラクタ・コード
E6B6	1	コントロールコードをCRTに出力するフラグ（リテラル・モード）
E6B7	1	未使用
E6B8	1	ファンクションキー表示フラグ
E6B9	1	テキストモード（カラー…FF,B/W…0）
E6BA	1	コピーモード（ノーマル…FF，テキスト…FD，グラフィック…FA）
E6BB	2	EDIT 1番上の行番号
E6BD	2	EDIT 一番下の行番号
E6BF	1	8251エラー発生フラグ
E6C0	1	PORT 30Hに出力したデータ
E6C1	1	PORT 40Hに出力したデータ
E6C2	1	PORT 31Hに出力したデータ
E6C3	1	PORT E4Hに出力したデータ
E6C4	2	テキストVRAMアドレスTOP（初期値：F3C8）
E6C6	1	(LINE) INPUT WAIT フラグ
E6C7	1	ON TIME$ カウンティングフラグ
E6C8	1	未使用
E6C9	1	Communication Buffer Overflow フラグ
E6CA	1	イベントキーフラグ（^O,^S,^C）
E6CB	2	キューテーブルアドレス
E6CD	1	キー入力サプレスフラグ
E6CE	1	ファンクションキー　ストア中フラグ
E6CF	1	キー入力　リピートカウンタ
E6D0		キーステータステーブル1（キー入力ポートデータのCPL）
E6DC		キーステータステーブル2（キー入力　押している所=1）
E6E8	1	カセット使用中フラグ（WRITE…FF,READ…1）
E6E9	2	COPY文　ドット対応グラフィック出力値カウンタ
E6EB	1	LPT OPEN フラグ
E6EC	1	COM1のWIDTH
E6ED	1	RS-232Cの使用中フラグ（8251用コマンド）
E6EE	1	ON割り込みの全部の個数
E6EF	2	ON割り込みテーブルアドレス
E6F1	1	ON割り込みフラグ（カウンタ）
E6F2	16	ファンクションキー1のデータ
E702	16	ファンクションキー2のデータ
E712	16	ファンクションキー3のデータ
E722	16	ファンクションキー4のデータ
E732	16	ファンクションキー5のデータ
E742	16	ファンクションキー6のデータ
E752	16	ファンクションキー7のデータ
E762	16	ファンクションキー8のデータ
E772	16	ファンクションキー9のデータ
E782	16	ファンクションキー10のデータ
E792	16	ターミナルモードの時のファンクションキー6
E7A2	16	ターミナルモードの時のファンクションキー7
E7B2	16	ターミナルモードの時のファンクションキー8
E7C2	16	ターミナルモードの時のファンクションキー9
E7D2	16	ターミナルモードの時のファンクションキー10
E7E2	3	LINE文　サブルーチン　ジャンプテーブル1
E7E5	3	LINE文　サブルーチン　ジャンプテーブル2
E7E8	2	CLEARできる上限（E5FF）

アドレス	バイト数	機　　能・用　　途
E7EA		割り込み処理ルーチン E7EA…RS-232C　エントリ E808…VRTC　エントリ E80E…CLOCK　エントリ E814…USER　エントリ E81A…DISK1　エントリ E820…DISK2　エントリ
E826		モニタ用ルーチン
E850	1	変数名3文字目以降の長さ
E851		変数名バッファ（3文字目以降）
E877	2	配列変数テキストポインタ退避
E879		（未使用）':'が入っている
E87A		中間言語バッファ（最後の数バイトは使用されない）
E9B8		（未使用）
E9B9		行入力バッファ
EABC	1	変数領域ルーチン中でDIM文から呼ばれたことを示す
EABD	1	FACのタイプ
EABE	1	中間言語←→リストルーチンDATA文,"…"の中など中間言語に変換しないときに立つフラグ
EABF	1	中間言語←リスト、数字→行番号　フラグ
EAC0	2	RST 10H テキストポインタ
EAC2	1	RST 10H 読んだ文字
EAC3	1	RST 10H 用 FACのタイプ
EAC4	8	RST 10H 用 FAC
EACC	2	文字列エリアの始まりのアドレス
EACE	2	ストリングスタックポインタ
EAD0	30	ストリングスタック（3バイト×10組）
EAEE	3	テンポラリストリングディスクリプタ
EAF1	2	フリーエリアの終わりのアドレス
EAF3	2	Work
EAF5	2	Work（ガベージ　コレクション）
EAF7	2	FOR文テキストポインタ退避
EAF9	2	DATA文エラー行番号
EAFB	2	認識すべき変数のタイプ（変数認識ルーチン）
EAFC	2	READ/INPUTフラグ、USINGルーチンフラグ
EAFD	2	Work
EAFF	1	行番号→行アドレスの変更があったことを示すフラグ
EB00	1	AUTOモードフラグ
EB01	2	AUTOモードで次に発生させる行番号
EB03	2	AUTOモード増分
EB05	2	1文実行前のテキストポインタの値（RESUME文の為）
EB07	2	1文実行前のスタック（エラーが起こったときスタックをもどすため）
EB09	2	ERLの値
EB0B	2	エラーを起こしたときのテキストポインタの値
EB0D	2	ON ERROR GOTOのとび先行のアドレス
EB0F	1	エラートラップルーチン中フラグ（FF＝トラップルーチン）
EB10	2	Work
EB12	2	実行停止時（END, STOP）のときの行番号
EB14	2	CONT実行再開文のアドレス
EB16	2	ラベル領域開始アドレス
EB18	2	テキストエンドアドレス+1

アドレス	バイト数	機　　能・用　　途
EB1A	2	ラベル登録済みフラグ
EB1B	2	単数変数領域開始アドレス
EB1D	2	配列変数領域開始アドレス
EB1F	2	フリーエリアの始まりアドレス（変数領域の最後+1）
EB21	2	READ文データポインタ
EB23		変数の暗黙型（DEFINT, DEFSTRなどによって変更される）
EB3D	2	FN引数スタックポインタ
EB3F	2	FN引数テーブル1の長さ
EB41		FN引数テーブル1
EBA5	2	FN引数スタックポインタを指す
EBA7	2	FN引数テーブル2の長さ
EBA9		FN引数テーブル2
EC0D	1	FN引数テーブル1サーチ中フラグ（変数認識ルーチン）
EC0E	2	単純変数/FN引数サーチ終わりアドレス+1（変数認識ルーチン）
EC10	1	FNテーブルもサーチしなさいというフラグ(FN実行中のフラグ)(変数認識ルーチン)
EC11	2	Work
EC13	1	FNネスティングレベル
EC15	1	INPUT文でデータの数をかぞえるために空読みするフラグ
EC16	2	NEXT文アドレス
EC18	1	NEXT文フラグ（0：FOR文からきたことを示す。FOR文でNEXT文の処理ルーチンを使うから。）
EC19	4	FOR文初期値
EC1D	2	NEXT文行番号
EC1F	1	OPTION BASE文の値
EC20	1	OPTION BASE文実行済フラグ
EC21	1	中間言語→リストのスペースコントロール、CALL文Work
EC22	1	INPUT文での'?'を出力するかどうかのフラグ、CALL文Work
EC23	2	CHAIN文　（EAF1, F2）の退避
EC25	2	未使用
EC27	1	暗号化セーブ中であることを示す。
EC28	1	平文化ロード中であることを示す。
EC29	1	プログラム　プロテクトフラグ
EC2A	1	CHAIN文　MERGEフラグ
EC2B	1	CHAIN文　DELETEフラグ
EC2C	2	CHAIN文　DELETEエンドアドレス
EC2E	2	CHAIN文　DELETEトップアドレス
EC30	1	CHAIN文　フラグ
EC31	2	CHAIN文　実行行番号
EC33	8	SWAP用FRG（Floatingpoint Register）
EC3B	1	TRONフラグ
EC3C		倍精度で計算中に使用する最下位バイト
EC3D		FAC.　　　　倍精度仮数部最下位
EC3E		FAC.　　　　倍精度仮数部
EC3F		FAC.　　　　倍精度仮数部
EC40		FAC.　　　　倍精度仮数部
EC41		FAC.　整数型下位、単精度仮数部下位、倍精度仮数部
EC42		FAC.　整数型上位、単精度仮数部中位、倍精度仮数部
EC43		FAC.　　単精度仮数部上位、倍精度仮数部最上位
EC44		FAC.　　単精度指数部、　倍精度指数部
EC45	1	FAC演算用フォーマットでの符号
EC46	1	OV,/0エラーフラグ（演算）

アドレス	バイト数	機　能・用　途
EC47	1	OV フラグ（文字列→数値）
EC48	1	絶対値<1のとき非指数形式にするフラグ
EC49	1	計算中に使用する最下位バイト
EC4A EC51	7 1	倍精度用 FRG 仮数部 倍精度用 FRG 指数部
EC52		数値を文字列にするときのバッファ
EC6A		未使用
EC6D		倍精度除算用 Work
EC75		未使用
EC7A		単精度乗算用 Work
EC7D	1	接続されているドライブ数
EC7E	1	オープンできるファイル数
EC7F	2	ファイルバッファアドレステーブルのアドレス
EC81	2	ドライブポインタのアドレス
EC83	2	ヌルバッファアドレス
EC85	1	カレント　ドライブ番号
EC86	2	カレント　ドライブテーブルのアドレス
EC88	2	カレント　ファイルバッファのアドレス
EC8A EC8C EC8D	2 1 1	ディレクトリサーチ時のファイル名ポインタ (EF8A, 8B)のセクタ中のファイル番号 (EF8A, 8B)のセクタ番号
EC8E	1	ファイルモード（OPEN）LOAD, R のフラグ
EC8F EC98	9 9	ファイル名1 ファイル名2
ECA1 ECA2	1 1	DSKO$, DSKI$ トラック番号 DSKO$, DSKI$ セクタ番号
ECA3	1	bit7：すべてのファイルを close しない bit0：ヌルバッファは close しない
ECA4	1	SAVE 中であることを示す
ECA5		ディスクパラメータ ECA5　最大トラック番号 ECA6　1トラックあたりのセクタ数 ECA7　片面=0，両面=1 ECA8　1トラックあたりのクラスタ数
ECA5		ECA9　ボリュームあたりのクラスタ数 ECAA　ディレクトリトラック番号 ECAB　1クラスあたりのセクタ数 ECAC　FAT の関始セクタ番号 ECAD　FAT の終了セクタ番号 ECAE　FAT の数 ECAF　ディスク属性の入っているセクタ番号 ECB0　未使用（1クラスタあたりのセクタ数×2）
ECB1	2	BSAVE END アドレス
ECB3	1	未使用
ECB4 **ECB5**	1 1	DISK R/W エラー回数（N88DISK-BASIC では使われない） DISK R/W エラー回数（4回になると DISK I/O エラー）
ECB6	1	リードアフターライトフラグ
ECB7 ECB8	1 1	EBCDIC コードフラグ (ECB7)退避
ECB9	2	未使用
ECBB		フックアドレステーブル（別表）

アドレス	バイト数	機　　能・用　　途
EE0E		I/O 処理ルーチンジャンプテーブル（別表）
EE71		DISK 命令ジャンプテーブル（別表）
EECB		ON 割り込みジャンプテーブル（別表）
EF0A	1	ROM3をコールしたときの Acc, リターンしたときの Acc
EF0B	2	ROM3をコールしたときの HL
EF0D	1	COPY4,5のフラグ（COPY4…2, COPY5…3）
EF0E	1	ポート E6H に出力したデータ（インタラプトマスク）
EF0F	1	DMA FD 使用中フラグ
EF10	1	FD イニシャライズ中フラグ
EF11	1	DMA FD イニシャライズ中0になる
EF12	2	論理転送アドレス
EF14	1	ミニ FD TIME OUT までの時間
EF15	1	ミニ FD 高速転送フラグ
EF16	1	DMA FD 転送ルーチン中は FF（BUSY）
EF17	1	DMA FD R/W スイッチ（WRITE =2, READ =3, INIT =0）
EF18	1	DMA FD ドライブ番号、サーフェス番号
EF19	1	DMA FD トラック番号
EF1A	1	DMA FD セクタ番号
EF1B	1	DMA FD セクタ数
EF1C	2	DMA FD 物理転送アドレス
EF1E	3	DMA FD タイマ
EF21	1	DMAC へのコマンド（READ =40H, WRITE =80H）
EF22	1	FDC へのコマンド（READ =46H, WRITE =45H）
EF23	1	DMA FD エラー番号（READ =87H, WRITE =86H）
EF24	1	DMA FD リトライ減算カウンタ
EF25	1	未使用
EF26	1	ステータス0　FDC Result phase で読んだ値
EF27	1	ステータス1　FDC Result phase で読んだ値
EF28	1	ステータス2　FDC Result phase で読んだ値
EF29	1	シリンダ番号　FDC Result phase で読んだ値
EF2A	1	ヘッド（サーフェス）番号　FDC Result phase で読んだ値
EF2B	1	セクタ番号　FDC Result phase で読んだ値
EF2C	1	セクタ長　FDC Result phase で読んだ値
EF2D	8	DMA FD ドライブ ステータス テーブル1（2バイト×4）
EF35	8	DMA FD ドライブ ステータス テーブル2（2バイト×4）
EF3D	1	DMA FD エラー番号
EF3E	1	インタラプト ウェイト フラグ
EF3F	1	割り込みレベル退避
EF40	1	FDC INT ステータス（ST0）
EF41	8	マージンデータテーブルポインタ（2バイト×4）
EF49	1	BLOAD の R フラグ
EF4A	1	R/W セクタ数
EF4B	2	＜EF4B ～ EF5C は DMA FD 用の定数＞ ドライブ ステータス テーブル アドレス
EF4D	2	マージンコントロール ポートアドレス
EF4E	1	インタフェースボードチェックポートアドレス（bit0=0…あり）
EF4F	1	モーターコントロールポートアドレス（PRECOMPENSATION のため）
EF50	1	最大シリンダ番号
EF51	1	まん中のシリンダ番号（このシリンダ以上では PRECOMPENSATION を使う）

アドレス	バイト数	機　能・用　途
EF52	1	1トラックあたりのセクタ数
EF53	1	GAP3の長さ（未使用）
EF54	1	FDC ステータスポートアドレス
EF55	1	FDC データポートアドレス
EF56	1	FD レディデータ（5インチ=10H,8インチ=20H）F3H に OUT する
EF57	1	Step rate time
EF58	1	Head load time
EF59	1	DMAC モードデータ
EF5A	1	DMAC チャネルアドレス
EF5B	1	DMAC ベースアドレス
EF5C	1	インタフェースイネーブルデータ．F3H に OUT する
EF5D	1	カレントドライブタイプ（0,1= DMA，2=ミニ片面，3=ミニ両面）
EF5E	1	未使用
EF5F	1	各ディスクタイプごとのドライブ番号
EF60	1	DMA 8 インチのドライブ数
EF61	1	DMA 5 インチのドライブ数
EF62	1	ミニ FD のドライブ数
EF63	1	ミニ FD 両面/片面（片面=0，両面=80H）［PC −8801mk Ⅱでは両面］
EF64		ドライブタイプ対応表（ドライブ番号→ドライブタイプ）
EF70	1	未使用
EF71	1	TERM センドイネーブル
EF72	1	TERM HALF/FULL
EF73	2	TERM テキスト・ポインタ退避
EF75	1	TERM リテラルフラグ（f・6）
EF76	2	TERM ROLL バッファの最後を指す
EF78	2	TERM ROLL データ（画面より上）の最後を指す
EF7A	2	TERM ROLL バッファの最初を指す
EF7C	2	TERM ROLL データ（画面より下）の最後を指す
EF7E	1	TERM ROLL バッファがあるかないかを示す（1のときある）
EF7F	1	PORT 30H（IN）の CPL（ディップスイッチ1）
EF80	1	PORT 31H（IN）の CPL（ディップスイッチ2）
EF81	1	ライトペン Y 座標
EF82	1	ライトペン X 座標
EF83	2	1行入力最初の画面上位置
EF85	1	1行入力最後の桁
EF86	1	カーソル位置 Y（1〜）
EF87	1	カーソル位置 X（1〜）
EF88	1	画面縦
EF89	1	画面横
EF8A	2	アトリビュートアドレス
EF8C	1	セットするアトリビュートコード
EF8D	1	アトリビュート桁
EF8E	1	アトリビュート桁カウンタ
EF8F	1	CRT に表示した文字
EF90	1	ファンクションキー表示開始番号（0または5）
EF91	8	COPY GVRAM コピー用バッファ
EF99		未使用
EF9A		テキスト画面行縦続コード（0：つながっている、0AH：^J、その他：切れている）
EFB3	1	未使用
EFB4	1	エディットモードフラグ
EFB5	2	行サーチ、テンポラリアドレス

アドレス	バイト数	機　能・用　途
EFB7	2	行サーチ、サーチした行の1つの前の行のアドレス
EFB9	1	01=エディット、FF=1行入力、00=ノーマル
EFBA	2	エラー位置を決めるため、テキストリードルーチンでHLの位置をSAVEしておく。
EFBC	2	エラー位置（HELPでカーソルが移動するところ）
EFBE	2	エラーのあったところの行番号
EFC0	2	中間コード→リスト形式にしたときの（EFBC）に対応するエラー位置
EFC2	2	エラー行をリストしたときのエラー位置のカーソル座標
EFC4	2	（LINE）INPUT WAIT文 待ち時間
EFC6	3	（LINE）INPUT WAIT用 減算カウンタ（0.1秒単位）
EFC9		ON TIME$用減算カウンタ（2秒単位）
EFCD		キューテーブル（6バイト×2組）EFCD～D2はキー入力バッファ用 EFCD EFD3…Putオフセット CE　D4…Getオフセット CF　D5…Back Character D0　D6…キューの長さ（2^n-1） D1, D2 D7, D8…キューアドレス
EFD9	32	キー入力バッファ
EFF9	1	キースキャンフラグ（3=押していない、FF=押したまま、2=新しくキーを押した）
EFFA	1	キースキャン 新しく押されたキーのビットが1
EFFB	2	キーステータステーブル2（E6DC～E6E7）用ポインタ
EFFD	1	キーのコード
EFFE	1	入力したキーのASCIIコード
EFFF	1	キー入力シフトキーのポートデータ
F000	1	前回シフトキーのポートデータ
F001	1	CAPS LOCKフラグ
F002	1	シフト番号(0=ノーマル, 1=SHIFT, 2=CTRL, 3=カナ, 4=カナSHIFT, 5=GRPH)
F003	2	ファンクションキーアドレス
F005	1	カセット ファイル属性
F006	1	カセット テープリードエラー禁止フラグ（ヘッダサーチ中）
F007	1	カセットSTOPキーでNEWしないフラグ（ロード中は0：NEWする）
F008	1	カセットBack character
F009	1	カセット ボーレート（FB=1200ボー, FA=600ボー）
F00A	1	カセット ベリファイフラグ
F00B	1	TERM Xパラメータ有効フラグ&ステータス（^S, ^Q）
F00C	1	ファンクションキー番号
F00D	6	時間用バッファ F00D 秒（BCD） F00E 分（BCD） F00F 時（BCD） F010 日（BCD） F011 月（バイナリー） F012 年（BCD）
F013	1	PUT@文 条件またはフォアグラウンド・バックグラウンドがあるかどうかのフラグ
F014	1	PUT@文 フォアグラウンドカラーパレット番号
F015	1	PUT@文 バックグラウンドカラーパレット番号
F016	1	PUT@文 フォアグラウンドPUT用マスク
F017	1	PUT@文 バックグラウンドPUT用マスク
F018	1	PUT@文Work（(F014)のコピー）
F019	1	PUT@文Work（(F015)のコピー）
F01A	2	スクリーン座標X
F01C	2	スクリーン座標Y

アドレス	バイト数	機　能・用　途
F01E	1	フォアグラウンドカラー パレット番号（COLOR文）
F01F	1	バックグラウンドカラー パレット番号（COLOR文）
F020	1	ボーダーカラーのカラーコード（PC－8801mkⅡでは無意味）
F021	2	GET@文 パターンの横のドット数、LINE文 横のドット数
F023	2	GET@文 パターンの縦のドット数、LINE文 縦のドット数
F025	2	LINE文 ラインスタイル
F027	2	LP（Last Referenced Point）のスクリーン座標X
F029	2	LPのスクリーン座標Y
F02B	2	ビューポート左X
F02D	2	ビューポート右X
F02F	2	ビューポート上Y
F031	2	ビューポート下Y
F033	2	GVRAMのリード/ライトするメモリのアドレス
F035	1	上記のメモリの中でのビット位置（マスクパターン）
F036	1	GVRAM0のマスクパターン
F037	1	GVRAM1のマスクパターン
F038	1	GVRAM2のマスクパターン
F039	2	1つのGVRAMあたりの縦の座標の最大（＝199）
F03B	2	GET@，PUT@文 配列データポイント
F03D	1	GET@，PUT@文 横のバイト数
F03E	1	GET@，PUT@文 最右バイト中のドット位置
F03F	1	GET@，PUT@文 最左バイト中のドット位置
F040	1	GET@，PUT@文 最左バイト用マスク
F041	1	GET@，PUT@文 最右バイト用マスク
F042	2	GET@文 サブルーチンアドレス（条件によって変わる）
F044	1	PAINT文 領域色パレット番号、LINEの色パレット番号
F045	1	PAINT文 境界色パレット番号
F046	1	境界色によるGVRAM0用マスク
F047	1	境界色によるGVRAM1用マスク
F048	1	境界色によるGVRAM2用マスク
F049	1	PAINT文 行き止まり判断用フラグ
F04A	1	未使用
F04B	1	アクティブページ（COPY文、PUT@文などのWork）
F04C	2	ペアレジスタ退避用
F04E	1	レジスタ退避用
F04F	2	PAINT文 タイルストリングの先頭のアドレス
F051	1	PAINT文 タイルストリングとバックグラウンドストリングが等しいことを示す。
F052	2	PAINT文 タイルストリングの終わりのアドレス+1
F054	2	PAINT文 バックグラウンドストリングの先頭のアドレス
F056	1	PAINT文 タイルペイントを行なうかどうかのフラグ
F057	1	（タイルストリングの長さ）￥M－1（M：カラーモード3，B/Wモード1）
F058	1	PAINT文 タイルストリングカウンタ
F059	3	バックグラウンドストリングデータ（8ドット分）
F05C		未使用
F062	2	PAINT文 で使えるフリーエリアの大きさ
F064	2	PAINT文 サーチポイント・キューの長さ
F066	2	PAINT文 サーチポイント・キューの終わり
F068	2	PAINT文 サーチポイント・キューの先頭
F06A	1	PAINT文 サーチ方向
F06B	2	PAINT文 右へペイントしたドット数
F06D	2	PAINT文 前回ペイントしたドット数

アドレス	バイト数	機　　能・用　　途
F06F	1	PAINT 文 左へペイントしたかどうかのフラグ
F070	1	PAINT 文 右へペイントしたかどうかのフラグ
F071	1	GET@, PUT@のフラグ (GET@…0, PUT@…80H)
F072	2	CIRCLE 文 (楕円率) *256 (比率=1…100H, 比率=0.5 or 2…80H)
F074	2	CIRCLE 文 (開始角度, 終了角度のうち小さい方の値) ×90　(未指定0000)
F076	2	CIRCLE 文 (開始角度, 終了角度のうち大きい方の値) ×90+1 (未指定FFFF)
F078	2	CIRCLE 文 Work
F07A	1	開始角度、終了角度の大小を示すフラグ (開始>終了…FF、開始≦終了…00)
F07B	1	扇形を書くのかどうかのフラグ。1で書く。(ビット7…終了時 ビット0…開始時)
F07C	2	CIRCLE 文 半径× SQR(2)/2
F07E	2	CIRCLE 文 Work
F080	2	CIRCLE 文 円周上の点の X 座標 (中心に対して)
F082	2	CIRCLE 文 円周上の点の Y 座標 (中心に対して)
F084	1	CIRCLE 文 比率1以上かどうかのフラグ (1より大…01、1以下…00)
F085	2	CIRCLE 文 Work
F087	1	スクリーンモード (0,1,2)
F088	1	画面スイッチ (0,1,2,3)
F089	1	リード/ライト ページセレクトポート
F08A	1	ページ (カラーモード3, B/W モード1)
F08B	1	アクティブ ページ セレクトポート (カラーモードでは5C)
F08C	1	ディスプレイページ
F08D	2	縦のドット数 (640×400→400, 640×200→200)
F08F	1	ウインドウフラグ。WINDOW 文が実行されると1になる。
F090	2	ビューポート幅 (Sx2−Sx1)
F092	2	ビューポート丈 (Sy2−Sy1)
F094	4	ウインドウ幅 (Wx2−Wx1)
F098	4	ウインドウ丈 (Wy2−Wy1)
F09C	4	ビューポート幅/ウインドウ幅
F0A0	4	ビューポート丈/ウインドウ丈
F0A4	4	LP のワールド座標 X
F0A8	4	LP のワールド座標 Y
F0AC	4	グラフィック処理用 Work
F0B0	4	ウィンドウ 左X
F0B4	4	ウィンドウ 右X
F0B8	4	ウィンドウ 上Y
F0BC	4	ウィンドウ 下Y
F0C0	2	ビューポート (Sx1, Sy1) の GVRAM のメモリ・アドレス+80
F0C2	1	ビューポート (Sx1, Sy1) の GVRAM セレクトポート
F0C3	2	ビューポート (Sx1, Sy2) の GVRAM メモリアドレス
F0C5	1	ビューポート (Sx1, Sy2) の GVRAM セレクトポート
F0C6	2	ビューポート (Sx1, 0) のメモリアドレス
F0C8	2	ビューポート (Sx2, 0) のメモリアドレス
F0CA	1	ビューポート 左端の対応ビット
F0CB	1	ビューポート 右端の対応ビット
F0CC	2	PAINT 文 サーチするドットライン (ビューポート) の左端メモリアドレス
F0CE	2	PAINT 文 サーチするドットライン (ビューポート) の右端のメモリアドレス
F0D0	3	NEW ON 1で N-BASIC への切り換えに使う。
F0D3		カセット入力バッファ
F153	1	RS-232C SI/SO フラグ (入力)
F154	1	RS-232C SI/SO フラグ (出力)
F155		モニタ用ルーチン、ワークエリア

アドレス	バイト数	機　　能・用　　途
F21E		未使用
F300	32	インタラプトベクトル
F320		未使用
F3C8		テキスト VRAM
FF80	120	ヌルライン
FFF8		未使用

別表1　N88 DISK-BASIC 使用時のフックアドレステーブル

アドレス	内　　容
ECBB	JP 8A12　ディスク情報セット
ECBE	RET
ECC1	JP A736　何もしない
ECC4	JP 9BC5　ディスク SAVE ルーチン
ECC7	RET
ECCA	RET
ECCD	JP 8E58　ドライブテーブル INIT, ID 実行
ECD0	JP 9FFD　ディスク GET#/PUT#
ECD3	JP 91A5　EDIT行番号セット UP
ECD6	JP 91BE　EDIT 行番号セット DOWN
ECD9	JP 9C51　ディスク LOAD ルーチン
ECDC	JP A6E3　LINE INPUT# READ BACK
ECDF	RET
ECE2	JP 9118　ROLL DOWN キー
ECE5	JP 90CC　ROLL UP キー
ECE8	JP A08D　シーケンシャル出力
ECEB	RET
ECEE	JP A0D4　シーケンシャル入力
ECF1	RET
ECF4	RET
ECF7	RET
ECFA	RET
ECFD	RET
ED00	RET
ED03	RET
ED06	JP 8E55 デバイス#デフォルトセット
ED09	JP 8E9A GET デバイス#
ED0C	RET
ED0F	JP 8F0A シーケンシャル入力
ED12	RET
ED15	RET
ED18	RET
ED1B	RET
ED1E	RET
ED21	RET
ED24	JP 8DE5　1行入力後の処理
ED27	JP 8FA4 ディスク関数評価
ED2A	RET
ED2D	JP 8FBD ファイル関数評価

アドレス	内　　　容
ＥＤ３０	RET
ＥＤ３３	RET
ＥＤ３６	RET
ＥＤ３９	RET
ＥＤ３Ｃ	JP 8F06　LISTの先頭
ＥＤ３Ｆ	RET
ＥＤ４２	JP 91E3　1文字出力　カーソル補正
ＥＤ４５	RET
ＥＤ４８	RET
ＥＤ４Ｂ	RET
ＥＤ４Ｅ	RET
ＥＤ５１	RET
ＥＤ５４	RET
ＥＤ５７	RET
ＥＤ５Ａ	JP 8F7C　倍精度関数評価
ＥＤ５Ｄ	RET
ＥＤ６０	RET
ＥＤ６３	RET
ＥＤ６６	JP 8FDD　エラーメッセージセット
ＥＤ６９	RET
ＥＤ６Ｃ	RET
ＥＤ６Ｆ	JP 9B21 CLOSE　ディスクファイル
ＥＤ７２	RET
ＥＤ７５	JP 8F41　式の評価
ＥＤ７８	RET
ＥＤ７Ｂ	RET
ＥＤ７Ｅ	RET
ＥＤ８１	RET
ＥＤ８４	RET
ＥＤ８７	RET
ＥＤ８Ａ	RET
ＥＤ８Ｄ	RET
ＥＤ９０	RET
ＥＤ９３	RET
ＥＤ９６	RET
ＥＤ９９	RET
ＥＤ９Ｃ	RET
ＥＤ９Ｆ	JP AD48　プロテクトチェックⅠ
ＥＤＡ２	JP ACC1　SAVE暗号化
ＥＤＡ５	JP AD05　LOAD平文化
ＥＤＡ８	JP AD51　プロテクトチェックⅡ
ＥＤＡＢ	JP 9A01　OPENディスクファイル
ＥＤＡＥ	RET

アドレス	内　　　容
EDB1	JP 994B　ドライブポイントセット
EDB4	RET
EDB7	RET
EDBA	RET
EDBD	RET
EDC0	RET
EDC3	RET
EDC6	RET
EDC9	JP 8E4B　CHAIN キャンセル
EDCC	RET
EDCF	RET
EDD2	RET
EDD5	RET
EDD8	RET
EDDB	RET
EDDE	RET
EDE1	RET
EDE4	RET
EDE7	RET
EDEA	RET
EDED	RET
EDF0	RET
EDF3	RET
EDF6	JP 908B　ROM Ver1.0のためのパッチ
EDF9	RET
EDFC	RET
EDFF	JP 9977　マウント確認
EE02	RET
EE05	RET
EE08	RET
EE0B	RET

別表2　N88 DISK-BASIC 使用時のI/O 処理ルーチン　ジャンプテーブル

アドレス	内容	
EE0E	JP 4DC1　OPEN	
EE11	JP 4DC1　CLOSE	
EE14	JP 4DC1　PUT/GET	
EE17	JP 4DC1　OUTPUT	
EE1A	JP 4DC1　INPUT	
EE1D	JP 4DC1　LOC	COM2, 3
EE20	JP 4DC1　LOF	
EE23	JP 4DC1　EOF	
EE26	JP 4DC1　FPOS	
EE29	JP 4DC1　READ BACK	
EE2C	JP 4DC1　WIDTH#	
EE2F	JP 9043　OPEN	
EE32	JP 483D　CLOSE	
EE35	JP 905F　PUT/GET	
EE38	JP 9051　OUTPUT	
EE3B	JP 0B06　INPUT	
EE3E	JP 0B06　LOC	SCRN
EE41	JP 0B06　LOF	
EE44	JP 0B06　EOF	
EE47	JP 0B06　FPOS	
EE4A	JP 0B06　READ BACK	
EE4D	JP 0B06　WIDTH#	
EE50	JP 9001　OPEN	
EE53	JP 483D　CLOSE	
EE56	JP 9018　PUT/GET	
EE59	JP 0B06　OUTPUT	
EE5C	JP 900C　INPUT	
EE5F	JP 903C　LOC	KYBD
EE62	JP 0B06　LOF	
EE65	JP 0B06　EOF	
EE63	JP 0B06　FPOS	
EE6B	JP 9032　READ BACK	
EE6E	JP 0B06　WIDTH#	

別表 3　N88-DISK-BASIC 使用時の DISK 命令 ジャンプテーブル

アドレス	内　　容
E E 7 1	JP A72B　Disk I/O error
E E 7 4	JP A72E　Disk offline
E E 7 7	JP 8EC3　DSKF
E E 7 A	JP A98C　CHAIN
E E 7 D	JP 88E9　SEARCH
E E 8 0	JP A875　WHILE
E E 8 3	JP A89F　WEND
E E 8 6	JP AC75　WRITE
E E 8 9	JP A916　CALL
E E 8 C	JP 9DAD　SET
E E 8 F	JP 99B6　NAME
E E 9 2	JP 9BAD　KILL
E E 9 5	JP A185　DSKI$
E E 9 8	JP A1C4　DSKO$
E E 9 B	JP 9F2F　FILES
E E 9 E	JP 9F2A　LFIELS
E E A 1	JP 4DC1　WBYTE
E E A 4	JP 4DC1　RBYTE
E E A 7	JP 4DC1　POLL
E E A A	JP 4DC1　ISET
E E A D	JP 4DC1　IRESET
E E B 0	JP 4DC1　STATUS
E E B 3	JP 4DC1　STATUS
E E B 6	JP 4DC1　CMD
E E B 9	JP 4DC1　IEEE
E E B C	JP 8803　ROLL
E E B F	JP A7A8　BLOAD
E E C 2	JP A73D　BSAVE
E E C 5	JP AC72　COMMON
E E C 8	JP 89B3　ATN

別表4　N88 DISK-BASIC使用時のON割り込みアドレステーブル

アドレス	内　　容
EECB	ON STOP
EECE	ON COM1
EED1	ON COM2
EED4	ON COM3
EED7	ON PEN
EEDA	ON TIME$
EEDD	ON HELP
EEE0	ON KEY(1)
EEE3	ON KEY(2)
EEE6	ON KEY(3)
EEE9	ON KEY(4)
EEEC	ON KEY(5)
EEEF	ON KEY(6)
EEF2	ON KEY(7)
EEF5	ON KEY(8)
EEF8	ON KEY(9)
EEFB	ON KEY(10)
EEFE	
EF01	
EF04	
EF07	

17-1-3 モニタ

A）モニタ ROM 部分

アドレス	項　目　名	機　　　能
6000	データ	DB 'DE '
6002	モニタ　スタート	
6047	RST 38H	ブレークポイントからとんでくる。
6088	メインループ	
60F0	コマンド　テーブル	モニタ使用できるコマンド（22個）
6106	コマンドアドレス テーブル	各コマンドに対する処理アドレスのテーブル。 （2バイト×22組）
6135	*S コマンド	メモリ内容を書き換える。
61A3	*D コマンド	メモリ内容を出力する。
6254	*Xコマンド	CPU レジスタに関するコマンド。 CPU レジスタを変更する。…………6260 CPU レジスタを全て出力する。……62DA
63D8	*F コマンド	メモリ内容を定数で埋める。
6406	*G コマンド	ユーザー・プログラムを実行する。
6479	*I コマンド	入力ポートの値を読み込む。
6491	*O コマンド	出力ポートへデータを出力する。
64A6	*M コマンド	メモリの内容を転送する。
64D8	*E コマンド	スクリーンエディタでメモリ内容を変更する。 サブコマンドテーブル　6541～654B サブコマンドアドレステーブル　654C～6561 CTRL-f　65BD CTRL-b　65C4 up arrow　65CB down arrow　65EC right arrow　661B left arrow　662F ROLL UP　6649 ROLL DOWN　669C edit から抜ける。　66EE
6754	*W コマンド	メモリ内容をカセットテープにセーブする。
6806	*V,R コマンド	V コマンド……6806　カセットテープの内容をベリファイする。 R コマンド……6807　カセットテープからロードする。
695E	*B コマンド	8 進、16進の切り換えを行なう。
6976	*L コマンド	機械語をディスアセンブルする。
6D02	*A コマンド	入力行をアセンブルする。
6E5F	*^B コマンド	BASIC へ戻る。
6E7A	*P コマンド	プリンタスイッチを切り換える。
6E8A	プリンタ出力	画面に出力した行をプリンタに出力する。
6EEE	GET パラメータ	コマンド行からパラメータを得る。(HL レジスタ)
6F80	数値出力 I	1バイトの数値を出力する。(Acc)
6F9A	数値出力 II	2バイトの数値を出力する。(HL レジスタ)
6FA7	文字列出力	文字列を出力バッファに書き込む。
6FCF	BIN → ASCII	BIN コードを ASCII コードに変換する。
6FD7	HL, DE 比較	HL レジスタと DE レジスタとを比較する。
6F0D	1文字入力	キーボードから1文字入力する。
7025	小文字→大文字	小文字から大文字に変換する。
702E	文字列出力 I	(HL) から (HL) =0の前までを出力する。
7037	CR, LF 出力	CR, LF コードを出力する。

アドレス	項　目　名	機　　　　能
7044	スペース出力	スペース文字を出力する。
704C	文字列出力Ⅱ	（HL）から（HL）のビット7が1になる文字までを出力する。
7058	メッセージ	無意味なメッセージ！？
7129	1文字出力Ⅰ	Accの値を画面に出力する。
7159	キー入力	キーボードからの1文字入力を行なう。
7162	キーセンス	キーボードが押されているか調べる。（押されていればCY＝0）
716E	VRAM アドレス	カーソル位置から VRAM アドレスを得る。
717B	カーソル ON	カーソル表示を ON にする。
7183	カーソル OFF	カーソル表示を OFF にする。
718B	カセット WRITE ON	カセットインターフェイスへの書き込みを開始する。
7194	カセット出力	カセットインターフェイスへデータを出力する。（Acc）
719D	カセットWRITE OFF	カセットインターフェイスへの書き込みを終了する。
71A6	カセット READ ON	カセットインターフェイスからの読み出しを開始する。
71AF	カセット READ OFF	カセットインターフェイスからの読み出しを終了する。
71B8	カセット入力	カセットインターフェイスからデータを入力する。（Acc）
71C3	LINE INPUT	1行入力を行なう。
71CC	CRTC セット	CRTC（μPD3301）の設定を行なう。
71D5	BUSY チェック	プリンタからの BUSY 信号をチェックする。
71DA	1文字出力Ⅱ	プリンタへ1文字出力する。
71F4	データ	RAM 上（F155～F1CD）に転送されるサブルーチン。
7270	メインROM LDIR	メイン ROM をセレクトし、ブロック転送（LDIR）を行なう。
7276	メイン ROM LDDR	メイン ROM をセレクトし、ブロック転送（LDDR）を行なう。
727C	*TM コマンド	メモリのテストを行なう。
746F	HELP コマンド	コマンド一覧表を出力する。
77E8	Edit HELP	スクリーンエディタ時のサブコマンド一覧表を出力する。
7973	フラグ HELP	フラグをセットする時に、フラグの説明を出力する。

B）モニタ　ディスクコード部分（Aug. 19, 1983 バージョン）

アドレス	項　目　名	機　能
8400	ジャンプテーブル初期化	モニタコマンドのジャンプテーブル初期化
8427	^Dコマンド	フロッピィディスクの内容を表示する。
84F0	^Rコマンド	フロッピィディスクからデータをロードする。
84F1	^Wコマンド	フロッピィディスクにデータをセーブする。
8588	ディスク関係サブルーチン	^D, ^R, ^W, コマンド用のサブルーチンの集まり。
8622	サーフェス	^D, ^R, ^W, コマンドで、ディスクのサーフェスを表わす。
8623	拡張 HELP コマンド	HELP コマンドの拡張部分（ディスク用コマンドの説明を出力する）。
866A	メッセージ	HELP コマンド用のメッセージ
876A	M コマンド	M コマンドの完全バージョン
8790	ディスク情報セット	ディスク情報をワークエリアにコピーする。
8796	ドライブポインタのセット	ドライブポインタをセットする。

C）モニタ　ワークエリア部分

アドレス	項　目　名	機　　　能
E669	ブレークポイント用	ブレークポイント用ジャンプテーブル（RST38で使用）
E66C		イニシャライズ拡張用フック
E66F		拡張 M コマンド用フック
E672		^D コマンド・ジャンプテーブル
E675		^W コマンド・ジャンプテーブル
E678		^R コマンド・ジャンプテーブル
E67B		E コマンド拡張用フック
E67E		拡張 HELP コマンド用フック
E681		汎用フック
E826	モニタエントリ	モニタの BASIC からのエントリ
F155	N88-BASIC ROM コール	N88-BASIC ROM をコールする。 CALL　F155 DW　　ADRS
F16C	ROM セレクト	N-BASIC ROM のセレクトを行なう。
F184	ブロック転送	ブロック転送ルーチン LDIR…　F184 LDDR…F187
F18A	メモリアクセス	メイン ROM, RAM をセレクトし、メモリをアクセスする。
F1BC	エラールーチン	モニタ ROMをセレクトし、エラールーチンへ。
F1C2	GOコマンド用テーブル	GO コマンドでメイン ROMをセレクトしてジャンプする。
F1C8	ブレークポイント	RST38H でここへとんでくる。モニタ ROM をセレクトし、6047へ。
F1CE	RADIX フラグ	H or Q
F1CF	BREAK POINT フラグ	ブレークポイントの設定を示すフラグ（bit0, bit1）
F1D0	BP1 内容	ブレークポイント1にあった内容
F1D1	BP1アドレス	ブレークポイント1の設定アドレス
F1D3	BP2 内容	ブレークポイント2にあった内容
F1D4	BP2 アドレス	ブレークポイント2の設定アドレス
F1D6	D コマンドアドレス	
F1D8	S, E コマンドアドレス	
F1DA	L コマンドアドレス	
F1DC	A コマンドアドレス	
F1DE	コマンド保存	前に実行したコマンド
F1DF	E コマンドフラグ	プリンタへの出力をさせないため
F1E0		1バイト数値入力フラグ
F1E1	ROM セレクト	
F1E2	ファイル名1	
F1E8	ファイル名2	
F1EE		未使用
F1EF	READ/VERIFY フラグ	
F1F0		未使用
F1F1		2バイト数値入力フラグ
F1F2	オフセットアドレス	
F1F4	プリンタスイッチ	
F1F5	レジスタ退避	HL 退避
F1F7	レジスタ退避	SP 退避
F1F9	OAR 退避	テキスト ウィンドウ オフセットアドレスレジスタ退避
F1FA	フック退避	HOOK,（EDC9～）退避
F1FD		X コマンド用レジスタ退避
F217	数値変換バッファ	

17-1-4 コマンド・ステートメント・関数処理アドレス一覧表

```
J (( STATEMENT ))         MAIN  ROM4  DISK

AUTO ················ 0DD5  ----  ----
BEEP ················ 3EB4  ----  ----
BLOAD ··············· EEBF  ----  A7A8
BSAVE ··············· EEC2  ----  A73D
CALL ················ EE89  ----  A916
CHAIN ··············· EE7A  ----  A98C
CIRCLE ·············· 6ECE  79D6  ----
CLEAR ··············· 522E  ----  ----
CLOSE ··············· 4B04  ----  ----
CLS ················· 71B5  ----  ----
CMD ················· EEB6  ----  4DC1
COLOR ··············· 6EC6  6878  ----
COLOR@ ·············· ----  6927  ----
COLOR(1) ············ ----  6882  ----
COLOR(2) ············ ----  68EC  ----
COM ON/OFF/STOP ····· 7D71  ----  ----
COMMON ·············· EEC5  ----  AC72
CONSOLE ············· 7071  ----  ----
CONT ················ 5140  ----  ----
COPY ················ 6EA6  7053  ----
DATA ················ 0C77  ----  ----
DATE$ ··············· 721C  ----  ----
DEF ················· 15D7  ----  ----
  DEF FN ············ 15DB  ----  ----
  DEF USR ··········· 15C8  ----  ----
DEFDBL ·············· 0ACD  ----  ----
DEFINT ·············· 0AC7  ----  ----
DEFSNG ·············· 0ACA  ----  ----
DEFSTR ·············· 0AC4  ----  ----
DELETE ·············· 1B40  ----  ----
DIM ················· 5AC5  ----  ----
DSKO$ ··············· EE98  ----  A1C4
EDIT ················ 657B  ----  ----
ELSE ················ 0C79  ----  ----
END ················· 50E5  ----  ----
ERASE ··············· 519C  ----  ----
ERROR ··············· 0DCA  ----  ----
FIELD ··············· 4A5C  ----  ----
FILES ··············· EE9B  ----  9F2F
FOR ················· 08BF  ----  ----
GET ················· 7198  ----  ----
  GET# ·············· 4B42  ----  ----
  GET@ ·············· 71A2  7C33  ----
GOSUB ··············· 0BBF  ----  ----
GOTO ················ 0BF9  ----  ----
HELP ON/OFF/STOP ···· 72AB  ----  ----
IF ·················· 0E05  ----  ----
INPUT ··············· 102D  ----  ----
  INPUT ············· 1034  ----  ----
  INPUT# ············ 1020  ----  ----
  INPUT WAIT ········ 1034  ----  ----
```

《 STATEMENT 》	MAIN	ROM4	DISK
IRESET ·············	EEAD	----	4DC1
ISET ···············	EEAA	----	4DC1
KEY ················	6EFA	742E	----
KEY ··············	----	7443	----
KEY LIST ·········	----	73DF	----
KEY ON/OFF/STOP ··	----	7437	----
KILL ···············	EE92	----	9BAD
LET ················	0C9C	----	----
LFILES ·············	EE9E	----	9F2A
LINE ···············	0FAA	----	----
LINE ·············	6EAE	7E8B	----
LINE INPUT ·······	0FB9	----	----
LINE INPUT# ······	4CC1	----	----
LINE INPUT WAIT ··	0FB9	----	----
LIST ···············	18D9	----	----
LLIST ··············	18D4	----	----
LOAD ···············	4854	----	----
LOCATE ·············	714E	----	----
LPRINT ·············	0E4C	----	----
LSET ···············	49AB	----	----
MERGE ··············	4855	----	----
MID$ ···············	585A	----	----
MON ················	E826	----	----
MOTOR ··············	7F30	----	----
NAME ···············	EE8F	----	99B6
NEW ················	77DD	----	----
NEW ··············	4F00	----	----
NEW ON ···········	77E0	----	----
NEXT ···············	52BD	----	----
ON ·················	0D01	----	----
ON ERROR GOTO ····	0D05	----	----
ON GOSUB/GOTO ····	0D73	----	----
ON XX GOSUB ······	0D34	----	----
OPEN ···············	4798	----	----
OPTION BASE ········	1C89	----	----
OUT ················	17FA	----	----
PAINT ··············	6EDA	7674	----
PEN ON/OFF/STOP ····	72C8	----	----
POINT ··············	6EB2	6E25	----
POKE ···············	1B84	----	----
POLL ···············	EEA7	----	4DC1
PRESET ·············	6E96	7DFB	----
PRINT ··············	0E54	----	----
PSET ···············	6E9A	7E00	----
PUT ················	71A6	----	----
PUT# ·············	4B41	----	----
PUT@ ·············	71B0	7C33	----
RANDOMIZE ··········	1CD1	----	----
RBYTE ··············	EEA4	----	4DC1
READ ···············	10F9	----	----
REM ················	0C79	----	----
RENUM ··············	6F0E	75DD	----
RESTORE ············	50A5	----	----
RESUME ·············	0D8D	----	----
RETURN ·············	0C41	----	----
ROLL ···············	6ECA	EEBC	8803
RSET ···············	49AA	----	----
RUN ················	0B7C	----	----
SAVE ···············	48A3	----	----

SCREEN ·············	6EDE	698F	----
SET ················	EE8C	----	9DAD
STATUS ·············	EEB0	----	4DC1
STOP ···············	50CA	----	----
STOP ·············	50CD	----	----
STOP ON/OFF/STOP ·	72B0	----	----
SWAP ···············	515E	----	----
TERM ···············	7367	----	----
TIME$ ··············	7279	----	----
TIME$ ON/OFF/STOP	7279	----	----
TIME$= ···········	728F	----	----
TROFF ··············	5159	----	----
TRON ···············	5158	----	----
VIEW ···············	6EBA	6AC6	----
WAIT ···············	1800	----	----
WBYTE ··············	EEA1	----	4DC1
WEND ···············	EE83	----	A89F
WHILE ··············	EE80	----	A875
WIDTH ··············	181A	----	----
WIDTH ············	1875	----	----
WIDTH# ···········	1847	----	----
WIDTH LPRINT ·····	1866	----	----
WIDTH(DEV) ·······	1828	----	----
WINDOW ·············	6ED6	6C55	----
WRITE ··············	EE86	----	AC75

((FUNCTION))	MAIN	ROM4	DISK
ABS ················	20A0	----	----
ASC ················	5704	----	----
ATN ················	EEC8	----	89B3
ATTR$ ··············	----	----	9E1D
CDBL ···············	223E	----	----
CHR$ ···············	5714	----	----
CINT ···············	21A0	----	----
COS ················	2F8B	----	----
CSNG ···············	2214	----	----
CSRLIN ·············	3F31	----	----
CVD ················	4AC0	----	----
CVI ················	4ABA	----	----
CVS ················	4ABD	----	----
DATE$ ··············	6F02	----	----
DSKF ···············	EE77	----	8EC3
DSKI$ ··············	----	----	A185
EOF ················	4C51	----	----
ERL ················	13A2	----	----
ERR ················	1394	----	----
EXP ················	2E6E	----	----
FIX ················	2286	----	----
FN ·················	1600	----	----
FPOS ···············	4C62	----	----
FRE ················	58E4	----	----
HEX$ ···············	54C6	----	----
IEEE ···············	EEB9	----	4DC1
INKEY$ ·············	5AA3	----	----
INP ················	17E5	----	----
INPUT$ ·············	4BAC	----	A104
INSTR ··············	57D7	----	----
INT ················	2295	----	----
LEFT$ ··············	575A	----	----

LEN	56F8	----	----
LOC	4C2F	----	----
LOF	4C40	----	----
LOG	1F10	----	----
LPOS	1581	----	----
MAP	6E9E	6D29	----
MID$	5793	----	----
MKD$	4AA7	----	----
MKI$	4AA1	----	----
MKS$	4AA4	----	----
NOT	1512	----	----
OCT$	54C1	----	----
PEEK	1B7A	----	----
PEN	6EF2	72ED	----
POINT	6EC2	6DC0	----
POS	1586	----	----
RIGHT$	578A	----	----
RND	2F1A	----	----
SEARCH	EE7D	----	88E9
SGN	20B3	----	----
SIN	2F91	----	----
SPACE$	5741	----	----
SQR	2E05	----	----
STATUS	EEB3	----	4DC1
STR$	54CB	----	----
STRING$	5722	----	----
TAN	302C	----	----
TIME$	6EFE	752A	----
USR	158F	----	----
VAL	57B4	----	----
VARPTR	13B0	----	----
VIEW	6EE2	6CA8	----
WINDOW	6ED2	6CD5	----

17-2 サブシステム エントリー 一覧表

FDC765に関する略称

HD (Head)	物理的ヘッド番号0または1を指定します。
US1, US0 (Unit Select)	ドライブユニット番号0～3を示します。
C (Cylinder Address)	メディアのシリンダ番号0～76を示します。
H (Head Address)	論理的ヘッド番号を示します。
R (Record(Sector) Address)	セクタ番号を示します。
N (Record(Sector) Length)	1セクタ内のデータ長を示します。
EOT (End of Track)	あるトラック上の最終セクタ番号を示します。
GPL (Gap Length)	VFO SYNC を除いたGap3の長さ(バイト数)を示します。
DTL	処理すべきセクタあたりのデータ長
ST0, ST1, ST2 (Status)	リザルトステータス情報をストアするレジスタ(408ページ参照)
ST3 (Status)	ドライブの状態を示す情報をストアするレジスタ(410ページ参照)

その他の略称

Acc	アキュムレータ
SP	スタックポインタ
PC	プログラムカウンタ
PSTB	プリンタストローブ

〔注意：ドライブ番号は0，1，2，3になっています。〕

サブシステム ROM　0000H～07FFH

アドレス	項　目　名	内　　容
0000H	コールドスタート (JP 007EH)	リセットのとき、ここから始まる。
0008H	ブレークポイント処理	7F22Hをコールした後、004EHへ行く。 (7F22H番地参照)
0010H	1文字入力 (JP 06D9H)	ハンドシェイクによりメインシステムから1バイト入力する。CALL 0010HもしくはRST 10Hを実行すると、結果をAccに入れ、戻る。
0018H	1文字出力 (JP 070FH)	ハンドシェイクによりメインシステムへ1バイト出力する。データをAccに入れCALL 0018HもしくはRST 18Hを実行するとよい。
0020H ∫ 004DH	ジャンプテーブル	

アドレス	飛び先	内　　容
0024H	JP 06D9H	1文字入力
0027H	JP 070FH	1文字出力
002AH	JP 00C1H	ホットスタート
002DH	JP 02D5H	リード　データ
0030H	JP 02D6H	ベリファイ　データ
0033H	JP 0396H	ライト　データ
0036H	JP 0196H	リキャリブレイト
0039H	JP 01E7H	フォーマット（1トラックのみ）
003CH	JP 02C2H	フォーマット（全トラック）
003FH	JP 029AH	FDC765より1バイト入力
0042H	JP 02A7H	FDC765へ1バイト出力
0045H	JP 0263H	FDC765よりリザルトステータス入力
0048H	JP 0741H	高速ハンドシェイクによる2バイト入力
004BH	JP 077BH	高速ハンドシェイクによる2バイト出力

アドレス	項　目　名	内　　容
004EH	ブレークポイント処理	ブレークポイントにより実行を中断したときのアドレスレジスタをワークエリアに退避させ、中断したアドレスを上位、下位の順でメインシステムへ送る。 (RST 18Hを使用)
007EH	システム イニシャライズ	ワークエリア、8255のイニシャライズ。このイニシャライズが実行されると必ず片面指定に戻る。 8801/mkIIでは、STOP RESETすると再度、両面指定を行うようになっている。
00C1H	ホットスタート	メインシステムからのコマンド待ち。メインシステムからコマンドが送られてくると、そのコマンドの

アドレス	項　目　名	内　　　容
00C1H		処理番地をコールするようになっている。(SPは、00C1Hで8000Hに設定される。) また、ディスクドライブのモーターのON,OFFも行なっている。メインシステムからの指令が3分間以上なければ、モーターを止めるようになっている。
011BH ≀ 0158H	各コマンドの処理番地のデータ	

コマンドNo.	アドレス	内　　　容
0	015DH	ディスクのイニシャライズ
1	045EH	メインシステムから送られたデータをディスケットへ書き込む。
2	048CH	ディスケットからデータを読み込み、リードバッファにおく。
3	04DDH	リードバッファにあるデータをメインシステムへ送る。
4	0438H	セクタ単位のコピー
5	0517H	フォーマット
6	050BH	コマンドステータスをメインシステムへ送る。
7	0511H	ドライブステータスをメインシステムへ送る。
8	0649H	メモリテスト
9	0693H	メモリの内容をメインシステムへ送る。(アドレス指定)
10	0159H	I/OアドレスF7Hにデータを出力
11	052FH	メモリの内容をメインシステムへ送る。(転送数指定)
12	053BH	メインシステムからデータを受けとる。
13	056CH	指定したアドレスへジャンプさせる。
14	0571H	ディスケットからデータを読み、指定したアドレス上におく。
15	057FH	指定したアドレス上のデータをディスケットへ書き込む。
16	058EH	ドライブ0のトラック0セクタ1から1セクタ読み、5000Hにおく。そして5000Hへジャンプする。
17	045FH	メインシステムから送られたデータをディスケットへ書き込む。(高速ハンドシェイク使用)
18	04DEH	リードバッファにあるデータをメインシステムへ送る。(高速ハンドシェイク使用)
19	05A9H	FDC765のリザルト・フェイズの結果をメインシステムへ送る。(コマンド, ST0, ST1, ST2, C, H, R, Nの8バイト)
20	05B4H	指定したドライブのデバイスステータス(ST3)をメインシステムへ送る。
21	0547H	メモリーの内容をメインシステムへ送る。(高速ハンドシェイク使用、転送数指定)

<table>
<tr><th>アドレス</th><th>項　目　名</th><th>内　　　　容</th></tr>
<tr><td></td><td colspan="2">
<table>
<tr><th>コマンドNo.</th><th>アドレス</th><th>内　　　　容</th></tr>
<tr><td>22</td><td>0553H</td><td>メインシステムからデータを受けとる。(高速ハンドシェイク使用)</td></tr>
<tr><td>23</td><td>05BCH</td><td>両面・片面指定</td></tr>
<tr><td>24</td><td>05CEH</td><td>現在の両面・片面指定の状況を送る。</td></tr>
<tr><td>25</td><td>05D4H</td><td>リードアフターライトフラグ　セット</td></tr>
<tr><td>26</td><td>05D5H</td><td>リードアフターライトフラグ　リセット</td></tr>
<tr><td>27</td><td>05DAH</td><td>前回ブレークポイントで止まったアドレスから実行する。</td></tr>
<tr><td>28</td><td>0602H</td><td>ブレークポイントを設定</td></tr>
<tr><td>29</td><td>0628H</td><td>各レジスタの値を変える。(ブレークポイント用)</td></tr>
<tr><td>30</td><td>0629H</td><td>ブレークポイントで止まったときの各レジスタの値を送る。</td></tr>
</table>
</td></tr>
<tr><td>0159H</td><td>コマンド10</td><td>I/OアドレスF7Hにデータを出力する。(VFO回路用ウィンドウデータを変える。)</td></tr>
<tr><td>015DH</td><td>コマンド0
(イニシャライズ)</td><td>ディスクドライブの初期化
FDC765の初期値設定 (Head Unload Time, Step Rate Time, Head Load Time, Non DMA mode)
接続ドライブ数のチェックとリキャリブレート
(ヘッドをシリンダ0へ戻す。)</td></tr>
<tr><td>0196H</td><td>リキャリブレート</td><td>Cレジスタで指定 (C=0, 1, 2, 3) したドライブのヘッドをシリンダ0の位置へ戻す。エラーが起きたときはキャリーフラグを1にして戻る。
例　LD　C, 0
　　CALL　0196H
　　RET　NC
ドライブ1をリキャリブレートしてエラーがないならRETURNする。</td></tr>
<tr><td>01AAH</td><td>シーク</td><td>Cレジスタで指定したドライブのヘッドをDレジスタ(D=0～79) で指定したトラックまで移動させる。エラーが起きたときはキャリーフラグを1にして戻る。
例　LD　C, 0
　　LD　D, 10
　　CALL　01AAH
　　RET　NC
　　　⋮
ドライブ1のトラック10へヘッドを動かす。</td></tr>
<tr><td>01CCH</td><td>シークのエラーチェック</td><td>シークが正常終了しないか、実際に移動したトラック番号と要求したトラック番号が一致しなかったとき、キャリーフラグを1にしてリターンする。</td></tr>
<tr><td>01E7H</td><td>1トラックのみフォーマットする。</td><td>Cレジスタで指定したドライブのDレジスタで指定したトラックをフォーマットする。エラーが起きたときはキャリーフラグが1になる。</td></tr>
</table>

アドレス	項　目　名	内　　　容
0263H	リザルトフェイズ	リード・ライト・フォーマットコマンドなどのリザルトフェイズにおけるST0, ST1, ST2, C, H, R, Nの入力とエラーチェックを行う。エラーが起きたときはキャリーフラグを1にしてリターンする。 (Cレジスタ：ドライブNo)
029AH	リザルトフェイズのときにFDC765からデータを入力する。	結果はAccに入る。
02A4H	コマンドフェイズのときにFDC765へコマンドを出力する。	FDC765へコマンドを送るとき、コマンドデータをAccに入れCALL　02A4Hとする。
02A5H	コマンドフェイズのときにFDC765へデータを出力する。	FDC765へパラメータを送るとき、パラメータをAccに入れCALL　02A5Hとする。
02B4H	タイマー	
02C2H	全トラックのフォーマット	Cレジスタで指定されたドライブのディスケットをフォーマットする。エラーが起きるとキャリーフラグを1にして戻る。
02D5H	ベリファイ・データ	ディスケットからデータを読んだり、RAM上のメモリと比較したりする。
02D6H	リード・データ	
02FEH 031FH	リードモード ベリファイモード	Bレジスタ：リード・ベリファイするセクタ数（1～16） Cレジスタ：ドライブNo（0～3） Dレジスタ：トラックNo（1～79） Eレジスタ：セクタNo（1～16） HL：リード・ベリファイするRAMの先頭アドレス エラーのときはキャリーフラグを1にして戻る。
035BH	VFO用ウィンドウデータの初期化	Cレジスタで指定されたドライブのVFO用ウィンドウデータを初期化する。
0375H	VFO用ウィンドウデータの変更	ディスクのリード・ライトでエラーが起きるとき、ウィンドウデータを変えてリトライさせる。
0396H	ライト・データ	ディスケットへデータを書き込む。 Bレジスタ：書き込むセクタ数（1～16） Cレジスタ：ドライブNo（0～3） Dレジスタ：トラックNo（1～79） Eレジスタ：セクタNo（1～16） HLレジスタ：ライトするRAMの先頭アドレス エラーのときはキャリーフラグを1にして戻る。
03E9H	FDC765よりデバイスステータス（ST3）を入	Cレジスタにドライブナンバーを入れCALL　03E9Hを行なうと、Accにそのドライブのデバイスステータ

アドレス	項　目　名	内　　　容
03E9H	力する。	スが入る。
03F1H	FDC765へパラメータを送る。	ディスケットのリード・ライトで使うパラメータをFDCへ送る。(×××××HD US1 US0, C, H, R, N, EOT, GPL, DTLの順)
040EH	HD, US1, US0の計算	Cレジスタ：ドライブNo（0～3） Dレジスタ：トラックNo（0～79）より FDC765で使う形のヘッド・ユニット番号を計算する。 Acc＝ (bit 7～3: ×, 2: HD, 1: US1, 0: US0) サーフェス(ヘッド) {0…表, 1…裏} ドライブNo {0…ドライブ1, 1…ドライブ2, 2…ドライブ3, 3…ドライブ4}
041BH	シリンダナンバーの計算	Cレジスタ：ドライブNo（0～3） Dレジスタ：トラックNo（0～79）より シリンダナンバーを計算してAccに入れる。
0423H	両面・片面モードチェック	Cレジスタ：ドライブNo（0～3）より そのドライブが片面モードならAcc＝0、両面モードならAcc＝1を返す。
042FH	ヘッドナンバーの計算	Cレジスタ：ドライブNo（0～3） Dレジスタ：トラックNo（0～79）より そのトラックが表面にあるか裏面にあるかを計算する。 Acc＝ {0………表面, 1………裏面}
0438H	コマンド4 (COPY)	セクタ単位のコピーを行う。ただし2トラックにまたがったコピーはできない。(動作不定)
045EH	コマンド1 (Write data)	メインシステムから送られたデータをディスケットへ書き込む。(ライト・バッファを使用)
045FH	コマンド17 (Write data)	045EH：コマンド1の高速バージョン。 メインシステムとのハンドシェイクで高速ハンドシェイクを使っている。
048CH	コマンド2 (Read data)	ディスケットからデータを読み込みリードバッファ（5000H～5FFFH）におく。
04DDH	コマンド3 (Send data)	リードバッファにあるデータをメインシステムへ送る。転送量は、コマンド2のときに指定したセクタ数×256バイト
04DEH	コマンド18 (Send data)	04DDH：コマンド3の高速バージョン。 高速ハンドシェイクを用いている。

アドレス	項　目　名	内　　容
050BH	コマンド6 (Send command status)	コマンドステータスをメインシステムへ送る。 7: 1 / 6: FULL / 5〜1: × / 0: I/O ERR FULL: リードバッファに読み込んだデータがあるときには1，ないときは0 (ex. コマンド2を実行すると1になり コマンド3を実行すると0になる) I/O ERR: 処理中にI/Oエラーが起きたら1，そうでなければ0
0511H	コマンド7 (Send drive status)	ドライブステータスをメインシステムへ送る。 7: ドライブ＃3 / 6: ドライブ＃2 / 5: ドライブ＃1 / 4: ドライブ＃0 / 3: 1 / 2: 1 / 1: 1 / 0: 1 ドライブが接続されているときは1，そうでないときは0
0517H	コマンド5 (Format)	指定されたドライブの物理的フォーマットを行う。
052FH	コマンド11 (Send data)	メモリの内容をメインシステムへ送る。 先頭アドレスと転送バイト数を指定。
053BH	コマンド12 (Receive data)	メインシステムからデータを受けとり、指定されたアドレスへ置く。 先頭アドレスと転送バイト数を指定。
0547H	コマンド21 (Send data)	052FH：コマンド11の高速版 高速ハンドシェイクを使うのでデータは2バイトずつ送られる。転送バイト数は必ず2の倍数（偶数）にしなければならない。
0553H	コマンド22	053BH：コマンド12の高速版 高速ハンドシェイクを使うのでデータを2バイトずつ受けとる。転送バイト数は必ず2の倍数（偶数）にしなければならない。
055FH	コマンド21，22のサブルーチン	高速ハンドシェイク用に、先頭アドレスをHLに入れ、転送バイト数を½化してDEに入れる。
056CH	コマンド13 (Execute)	指定したアドレスへジャンプさせる。 このコマンドを実行するとリターンアドレス（00C1H＝Hot Start）をクリアしてジャンプするので、RET命令ではなくJP 002AH（JP 00C1H）を使うこと。
0571H	コマンド14 (Load data)	ディスケットからデータを読み、リードバッファではなく指定したアドレスへ置く。
057FH	コマンド15 (Save data)	ライトバッファではなく指定したアドレスのデータをディスケットへ書き込む。
058EH	コマンド16 (Load and go)	ドライブ0のトラック0、セクタ1から1セクタを5000Hに読み込み、5000Hへジャンプする。PC-80S31,

アドレス	項目名	内容
058EH		8031/2Wを単体でオートスタートさせるためのコマンドだが、ハードによる改造がないとオートスタートさせることはできない。
05A9H	コマンド19 (Send result status)	FDC765のリザルトフェイズの結果、コマンドステータスが、ステータス1(ST1), ステータス2(ST2), シリンダーNo(C), ヘッドNo(H), セクタNo(R), セクタ長(N)の8バイトをメインシステムへ送る。 ステータスバイトをみれば、どのようなエラーが起きたかわかる。(ex. CRCエラー)
05B4H	コマンド20 (Send device status)	指定したドライブのデバイスステータス(ST3)をメインシステムへ送る。主にライト・プロテクトのチェックに使われている。
05BCH	コマンド23 (Set up drive operation mode [Single/Double])	各ドライブの両面・片面指定を行なう。 メインシステムから送られるデータは次のとおり。 ビット7〜4: ×(未使用) ビット3: ドライブ#3 / ビット2: ドライブ#2 / ビット1: ドライブ#1 / ビット0: ドライブ#0 0………片面指定 1………両面指定
05CEH	コマンド24 (Send drive operation mode [Single/Double])	現在の両面・片面指定の状況をメインシステムへ送る。データはコマンド23(05BCH)のときと同じ。
05D4H	コマンド25 (Set read after write flag)	ディスケットへデータを書き込むコマンドで(コマンド1, 4, 15, 17)書き込み後､確認を行なうようにする。(リード･アフター･ライト)
05D5H	コマンド26 (Reset read after write flag)	リード･アフター･ライトモードを解除する。
05DAH	コマンド27 (Continue)	前回、ブレークポイントで止まったアドレスまたは、コマンド29で指定したPCのアドレスから実行させる。
0602H	コマンド28 (Set break point)	ブレークポイントを設定する。
0617H	ブレークポイント・クリア	ブレークポイントが設定してあれば元のデータに戻し、ブレークポイントのフラグをクリアする。
0628H	コマンド29 (Set new value to register)	各レジスタの値を変える。(ブレークポイント用) レジスタは番号で指定する。 AF — 0　HL' — 7 BC — 1　IX — 8

アドレス	項　目　名	内　　容
0628H		DE ― 2　IY ― 9 HL ― 3　I ― 10 AF′ ― 4　PC ― 11 BC′ ― 5　SP ― 12 DE′ ― 6
0629H	コマンド30 (Dump register value)	ブレークポイントで止まったときの各レジスタの値をメインシステムへ送る。 レジスタ番号は、コマンド29と同じ。
0649H	コマンド8 (Memory test)	4000H～7EFFHまで1バイトについて256回書き込みチェックを行なう。
0693H	コマンド9 (Send data)	メモリーの内容をメインシステムへ送る。 先頭アドレスと終了アドレス＋1を指定。
06A6H	メインシステムより4バイト入力	B, C, D, Eレジスタの順に入力
06A8H	メインシステムより3バイト入力	C, D, Eレジスタの順に入力
06AFH	メインシステムより2バイト入力	H, Lの順に入力
06B4H	トラックバッファモードチェック	トラックバッファモードなら（7F17Hが0でない）ドライブ、トラックNoが、前回アクセスしたものと同じかどうかチェックして、同じなら7F18Hに255を入れ、戻る。
06C9H	プリコンペイセイションのセット・リセット	ドライブNo（Cレジスタ：0～3）とトラックNo（Dレジスタ：0～79）よりシリンダNoを計算し、その結果シリンダNoが0～19なら7、20～39なら15をモーターコントロールポートF8Hへ出力する。つまり0～19ならプリコンペイセイションをセットして20～39ならリセットする。
06D9H	ハンドシェイクにより1文字入力	結果はAccに入る。
070FH	ハンドシェイクにより1文字出力	Accの内容を出力する。
0741H	高速ハンドシェイクにより2文字入力	(HL), (HL＋1)に入力。入力後HLはHL＋2されている。
077BH	高速ハンドシェイクにより2文字出力	(HL), (HL＋1)の内容を出力。出力後HLはHL＋2されている。
07B1H	ハンドシェイクにより2文字出力	(HL), (HL＋1)の内容を出力。出力後HLはHL＋2されている。 （1バイトのハンドシェイクを2回使用）

アドレス	項　目　名	内　　　容
07BCH	ハンドシェイクにより2文字入力	(HL), (HL+1) に入力。入力後HLはHL+2されている。 (1バイトのハンドシェイクを2回使用)
07C6H	PC-80S31, PC-8801mkII用のパッチルーチンI	FORMAT, WRITEコマンドでのパッチルーチン。動作終了後、待ち時間をおくようにしている。
07D6H	PC-80S31, PC-8801mkII用のパッチルーチンII	シーク・コマンドでのパッチルーチン。シーク終了後ステータス3 (ST3) のTwo sideビット (第2ビット) が1になるまで待つようにしている。
07EFH	デバイス・バージョンナンバー	FFH=PC-8031 EFH=PC-8031/2W, PC-80S31, PC-8801mkII 上位4ビットがデバイスナンバー、下位4ビットがソフトのバージョンを示す。
07F0H 〜 07FFH	VFO用ウィンドウ・データテーブル	

サブシステム ワーク・エリア　7F00H～7FFFH

アドレス	バイト数	内　容
7F00H	2	DRIVE＃0用のウィンドウデータ・テーブルのポインタ。
7F02H	2	DRIVE＃1用のウィンドウデータ・テーブルのポインタ。
7F04H	2	DRIVE＃2用のウィンドウデータ・テーブルのポインタ。
7F06H	2	DRIVE＃3用のウィンドウデータ・テーブルのポインタ。
7F08H	1	コマンド2（Read data）でリードしたセクタ数。 コマンド3（Send data）で使われる。
7F09H	1	モーターON/OFFフラグ（00H：OFF，FFH：ON）
7F0AH	1	リトライ・カウンター　エラーが起きたときFORMATなら5回、READ・WRITEなら9回のリトライが行なわれる。
7F0BH	1	現在のVFO用ウィンドウの値。
7F0CH	1	FDC765へ出力したコマンドの値。
（PC-8031/2Wのなごり。I/OアドレスF7Hは、プリンタポートとしても使われるため、プリンタへデータを出力した後、ウィンドウの値を戻しておかなければならなかった。）		
7F0DH	7	リザルト・フェイズで受けとったステータスの値。 7F0DH　ステータス0（ST0） 7F0EH　ステータス1（ST1） 7F0FH　ステータス2（ST2） 7F10H　シリンダ・ナンバー　（C） 7F11H　ヘッド・ナンバー　（H） 7F12H　セクタ・ナンバー　（R） 7F13H　セクタ長（N）
7F14H	1	コマンド・ステータス 7: DONE　6: FULL　5～1: （未使用）　0: I/O ERR I/O ERR……{処理中にI/Oエラーが起きたら1，そうでなければ0 FULL………{データ・アベイラブル・フラグ　リード・バッファに読み込んだデータがあるときは1，ないときは0（ex.　コマンド2を実行すると1になり　コマンド3を実行すると0になる。） DONE………{ラストI/Oコンプリート・フラグ　DISKがインテリジェント方式なので常に1

アドレス	バイト数	内　　容
7F15H	1	ドライブ・ステータス 7: ドライブ#3　6: ドライブ#2　5: ドライブ#1　4: ドライブ#0　3: 1　2: 1　1: 1　0: 1 ビット7〜4: ドライブが接続してあれば1, そうでなければ0 ビット3〜0: ドライブ・マウント・フラグだが常に1になっている。
7F16H	1	NON−ATN フラグ NON−ATN フラグ＝ { 00H：ハンドシェイクのATN信号を無視する。 / FFH：ハンドシェイクのATN信号を無視しない。 }
7F17H	1	トラック・バッファ・モードフラグ フラグ＝ { 00H：トラック・バッファ・モードでない / その他：トラック・バッファ・モード }
7F18H	1	前回リードしたドライブの番号（0〜254） もしFFHなら前回と同じドライブの同じトラックをリードしていることを示すフラグとなる。
7F19H	1	前回リードしたトラックの番号
7F1AH	1	前回リードしたセクタの番号。トラック・バッファ・モードでコマンド3を使ったときに使われる。
7F1BH	1	ドライブ・オペレーション・モード（両面/片面指定） 7〜4: 未使用（×）　3: ドライブ#3　2: ドライブ#2　1: ドライブ#1　0: ドライブ#0 { 0……片面指定 / 1……両面指定 }
7F1CH	4	各ドライブの両面/片面モード・フラグ　0のとき片面モード、それ以外（FFH）のとき両面モード。 7F1CH: ドライブ#0 7F1DH: ドライブ#1 7F1EH: ドライブ#2 7F1FH: ドライブ#3
7F20H	1	リード・アフター・ライト・フラグ (0 ——— リード・アフター・ライトを行なわない。 / それ以外 ——— リード・アフター・ライトを行なう。)
7F21H	1	リード、ベリファイ(リード・アフター・ライト用)の切り換えフラグ (0 ——— ベリファイ / それ以外 ——— リード)

アドレス	バイト数	内　　　　容
7F22H	3	RST8が実行されると（ブレーク・ポイント処理）ここをCALLするようになっている。 普通は7F22H：RETになっているが、ユーザーがJP命令に変えることで、ブレーク・ポイントの処理を自分でできるようになる。
7F25H	3	コマンド27（Continue）での実行先。 （7F25H：JP　××××）
7F28H	26	ブレーク・ポイント用レジスターの退避エリア（各2バイトづつ） 7F28H－SP　7F36H－BC' 7F2AH－PC　7F38H－AF' 7F2CH－I　7F3AH－HL 7F2EH－IY　7F3CH－DE 7F30H－IX　7F3EH－BC 7F32H－HL'　7F40H－AF 7F34H－DE'
7F42H	1	ブレーク・ポイント設定フラグ （0 ――― 設定されていない。 それ以外(FFH) ――― 設定されている。）
7F43H	2	ブレーク・ポイントを設定したアドレス
7F45H	1	ブレーク・ポイントを設定したアドレスの内容
7F46H ～ 7FFFH		スタック・エリア

FDC765 ステータスレジスタ、リザルトステータスの内容

（μPD765・7625ユーザーズマニュアル(NEC)より転記）

ステータスレジスタ

ビット番号	名　　称	略 称	内　　　容
D0	FD0 Busy	D0B	デバイス＃0がSEEKコマンドによるシーク動作を実行中であるか、シーク動作終了の割り込み要求を保留中であることを示します。
D1	FD1 Busy	D1B	デバイス＃1についてD0ビットの内容と同様。
D2	FD2 Busy	D2B	デバイス＃2についてD0ビットの内容と同様。
D3	FD3 Busy	D3B	デバイス＃3についてD0ビットの内容と同様。
D4	FDC Busy	CB	FDCがCommand Phase, Result Phase, またはリード／ライトコマンドのExecution Phaseを実行中であることを示します。このビットがセットされているときは、他のコマンドは受付けられません。
D5	Non-DMA MODE	NDM	FDCがNon-DMAモードでデータ転送中であり、メインシステムに対してサービスを要求していることを示します。
D6	Data Input/Output	DIO	データレジスタを介して転送するデータの方向を示します。0のときはメインシステムからFDCの方向、1のときはFDCからメインシステムの方向を示します。なお、データレジスタの状態はRQM(D7ビット)が示します。
D7	Request for Master	RQM	FDCからメインシステムへ転送すべきデータがデータレジスタにロードされていること、またはデータレジスタが空で、メインシステムからFDCへ転送するデータをデータレジスタに書き込んでもよいことを示します。データの方向を示すDIO(D6ビット)の状態により、次の働きをします。 DIO＝0のとき： 　メインシステムからFDCへデータを転送する場合で、メインシステムがFDCのデータレジスタにデータをセットしたとき（$\overline{\mathrm{WR}}$＝0）にRQMは0となり、FDCがそのデータを引き取ったとき1となります。 DIO＝1のとき： 　FDCからメインシステムへデータ転送する場合で、FDCがデータレジスタにデータをセットしたとき1となり、メインシステムがそのデータを引き取ったとき（$\overline{\mathrm{RD}}$＝0）0となります。

リザルトステータス0（ST0）

ビット番号	ステータス名称	略称	内容
D7	Interrupt Code	IC	INT要求が何によるかを示します。 D7D6 0 0　コマンドの正常終了（NT） 0 1　コマンドの異常終了（AT） 1 0　起動されたコマンドがInvalidであったため、コマンドを実行しなかった事を示します。（IC） 1 1　デバイスの状態遷移があった事を示します。（AI）
D6			
D5	Seek End	SE	SEEKまたはRECALIBRATEコマンドによるシーク動作が正常終了または異常終了したときにセットされます。
D4	Equipment Check	EC	デバイスからFault信号を受け取ったとき、またはRECALIBRATEコマンド時TRACK 0信号が一定時間内に検出できなかったときセットされます。
D3	Not Ready	NR	指定したデバイスがREADY状態でないときセットされます。
D2	Head Address	HD	INT要求時のヘッドの状態を示します。 SENSE INTERRUPT STATUSコマンド実行時は常に0となっています。
D1	Unit Select1	US1	INT要求時のデバイス番号を示します。
D0	Unit Select2	US0	

NT：Normal Terminate,　　AT：Abnormal Terminate
IC：Invalid Command,　　AI：Attention Interrupt

リザルトステータス1 (ST1)

ビット番号	ステータス名称	略 称	内 容
D7	End of Cylinder	EN	指定された最終セクタを越えてアクセスを継続させようとしたときセットされます。
D5	Data Error	DE	ディスク上のIDまたはデータを読み取った際、CRCエラーを検出するとセットされます。ID、データの区別はST2のDDビットによります。
D4	Over Run	OR	あるセクタのデータを処理しているとき、メインシステムのサービスが規定時間内に行われなかったときにセットされます。
D2	No Data	ND	1.次の4種のコマンド実行時にIDRで指定したセクタがトラック上で検出されなかったときセットされます。 (READ DATA, WRITE DATA, WRITE DELETED DATA, SCAN) 2.READ IDコマンド実行時トラック上にエラーのないIDが検出されなかったときセットされます。 3.READ DIAGNOSTICコマンド実行時セクタIDとIDRの内容が一致しなかったときセットされます。
D1	Not Writable	NW	WRITE DATA, WRITE DELETED DATA, WRITE IDコマンドを実行時に、書き込み不可状態を検出するとセットされます。
D0	Missing Address Mark	MA	1.ディスクのIDをアクセスするコマンドでIM(Index Mark)を2回検出するまでにAM(Address Mark)が検出されなかったときセットされます。 2.ディスクのデータを読み取るときDAMまたはDDAMが検出されなかったときセットされます。このときST2のMDビットもセットされます。

備考　D6とD3は使用されず常に0です。

リザルトステータス2（ST2）

ビット番号	ステータス名称	略 称	内 容
D6	Control Mark	CM	READ DATA またはSCAN コマンド実行時にDDAMの付いたセクタのデータを処理したときセットされます。
D5	Data Error in Data Field	DD	ディスクのデータを読み取るときCRC エラーを検出するとセットされます。
D4	No Cylinder	NC	ST1のNDビットに付帯したステータスで、IDのCバイトが一致しなかったときセットされます。
D3	Scan Equal Hit	SH	SCAN コマンドでEqual条件が満足されたときセットされます。
D2	Scan Not Satisfied	SN	SCAN コマンドで最終セクタまでスキャンしても条件に合うデータが検出されなかったときセットされます。
D1	Bad Cylinder	BC	ST1のNDビットに付帯したステータスで、IDのCバイトが一致せずFFHであったときセットされます。
D0	Missing Address Mark in Data Field	MD	ディスクのデータを読み取るときDAM， またはDDAMが検出されなかったときセットされます。

備考　D7は使用されず常に0です。

リザルトステータス3（ST3）

ビット番号	ステータス名称	略 称	内 容
D7	Fault	FT	デバイスからのFault信号の状態
D6	Write Protected	WP	デバイスからのWrite Protected信号の状態
D5	Ready	RY	デバイスからのReady信号の状態
D4	Track 0	T0	デバイスからのTrack 0信号の状態
D3	Two Side	TS	デバイスからのTwo Side信号の状態
D2	Head Address	HD	デバイスへのSide Select信号の状態
D1	Unit Select 1	US1	デバイスへのUnit Select 1信号の状態
D0	Unit Select 0	US0	デバイスへのUnit Select 0信号の状態

付録

付-1 機械語ルーチンソースリスト

付1-1 拡張PEEK, POKE

```
                ;--------------------------------
                ;      PC-TechKnow 8800mkII
                ;   <<  EXPANDED PEEK POKE   >>
                ;--------------------------------
                ;
                ;    PEEK ( <ADDRESS> [,<BANKNAME>] )
                ;    POKE <ADDRESS>,<DATA> [,<BANKNAME>]
                ;
                ;                           MODE
                ;  <BANKNANE> : NONE         5
                ;               " "          5
                ;               "T"          3
                ;               "N"          4
                ;               "n"        0,1,2
                ;

                       ORG   0E400H                ; OR 0DA00H

0B06            ERRIFC:EQU   0B06H                 ; ILLEGAL FUNCTION CALL
11D3            EVAL:  EQU   11D3H                 ; EVALUATE EXPR.
18A3            GTBYTE:EQU   18A3H                 ; GET PARAMETER(0-255) to Acc
1B93            GTADRS:EQU   1B93H                 ; GET ADRS to DE (-23768 to 65535)
56C9            CSFRFA:EQU   56C9H                 ; CHECK STRING & FREFAC

8450            SWAREA:EQU   8450H

E6C2            PORT31:EQU   0E6C2H                ; IMAGE OF PORT31H
EABD            FACTYP:EQU   0EABDH                ; TYPE of FAcc
EC41            FAC:   EQU   0EC41H                ; FAC ADRS

                ;-------- INITIALIZE --------
                ;
E400            INIT:
E400 3A74E5            LD    A,(WED27)
E403 B7                OR    A
E404 2034              JR    NZ,INIT9

E406 F3                DI
E407 2127ED            LD    HL,0ED27H
E40A 1174E5            LD    DE,WED27
E40D 010300            LD    BC,3
E410 EDB0              LDIR

E412 219FED            LD    HL,0ED9FH
E415 1177E5            LD    DE,WED9F
E418 010300            LD    BC,3
E41B EDB0              LDIR

E41D 3EC3              LD    A,0C3H
E41F 3227ED            LD    (0ED27H),A
E422 329FED            LD    (0ED9FH),A
E425 3A27ED            LD    A,(0ED27H)
E428 213CE4            LD    HL,PEEKQ
E42B 2228ED            LD    (0ED28H),HL
E42E 21EBE4            LD    HL,POKE
E431 22A0ED            LD    (0EDA0H),HL
E434 3E6C              LD    A,6CH
E436 3227E8            LD    (0E827H),A
E439 FB                EI
E43A            INIT9:
E43A FF                RST   38H
E43B C9                RET

                ;-------- PEEK --------
                ;
E43C            PEEKQ:                             ;from ED27
E43C 3C                INC   A
E43D 2804              JR    Z,PEEKQ2
```

```
E43F 3D                 DEC  A
E440 C374E5             JP   WED27
E443 23         PEEKQ2: INC  HL
E444 7E                 LD   A, (HL)
E445 FE97               CP   97H
E447 2804               JR   Z, PEEK
E449 2B                 DEC  HL
E44A 3EFF               LD   A, 255
E44C C9                 RET

E44D            PEEK:
E44D F1                 POP  AF
E44E CD77E5             CALL WED9F
E451 3E05               LD   A, 5
E453 F5                 PUSH AF                 ;SAVE MODE
E454 D7                 RST  10H
E455 CF                 RST  8H
E456 28                 DB   '('
E457 CD931B             CALL GTADRS
E45A D5                 PUSH DE                 ;SAVE ADRS
E45B 2B                 DEC  HL
E45C D7                 RST  10H
E45D FE29               CP   ')'
E45F 2809               JR   Z, PEEK10          ;GETEND
E461 CF                 RST  8H
E462 2C                 DB   ','
                ;      --- BANKNAME ---
E463 CDBCE4             CALL GETBN
E466 D1                 POP  DE                 ;ADRS
E467 C1                 POP  BC                 ;POP OLD MODE
E468 F5                 PUSH AF                 ;SAVE NEW MODE
E469 D5                 PUSH DE                 ;ADRS

E46A            PEEK10:
E46A CF                 RST  8H
E46B 29                 DB   ')'
E46C D1                 POP  DE                 ;ADRS
E46D ED5371E5           LD   (ADRS), DE
E471 F1                 POP  AF                 ;MODE
E472 3273E5             LD   (MODE), A
E475 E5                 PUSH HL                 ;TEXT POINTER
E476 3E02               LD   A, 2
E478 32BDEA             LD   (FACTYP), A        ;INTEGER
E47B 2A71E5             LD   HL, (ADRS)
E47E CD28E5             CALL IFSWAP
E481 CD99E4             CALL SETMDE
E484 4E                 LD   C, (HL)
E485 D35F               OUT  (5FH), A
E487 3EFF               LD   A, 255
E489 D371               OUT  (71H), A
E48B 3AC2E6             LD   A, (PORT31)
E48E D331               OUT  (31H), A
E490 FB                 EI
E491 69                 LD   L, C
E492 2600               LD   H, 0
E494 2241EC             LD   (FAC), HL
E497 E1                 POP  HL
E498 C9                 RET
                ;
                ;  ---- SET MODE ----
                ;
E499            SETMDE:
E499 F3                 DI
E49A 3A73E5             LD   A, (MODE)
E49D D603               SUB  3                  ; MODE >= 3 ?
E49F 3005               JR   NC, SMDE1          ; YES. NOT GVRAM
E4A1 3EFE               LD   A, 0FEH
E4A3 D371               OUT  (71H), A           ; SELECT 4th ROM#1
E4A5 C9                 RET
E4A6 2008       SMDE1:  JR   NZ, SMDE2
E4A8 3AC2E6             LD   A, (PORT31)
```

```
E4AB F602               OR    2
E4AD D331               OUT   (31H),A               ; SELECT TEXT RAM
E4AF C9                 RET
E4B0 3D         SMDE2:  DEC   A
E4B1 C0                 RET   NZ
E4B2 3AC2E6             LD    A,(PORT31)
E4B5 F604               OR    4
E4B7 E6FD               AND   0FDH
E4B9 D331               OUT   (31H),A               ; SELECT N-BASIC ROM
E4BB C9         SMDE3:  RET


                ;
                ;  ---- GET BANKMANE ----
                ;
E4BC            GETBN:
E4BC CDD311             CALL  EVAL
E4BF E5                 PUSH  HL                    ;TEXT POINTER
E4C0 CDC956             CALL  CSFRFA
E4C3 7E                 LD    A,(HL)
E4C4 B7                 OR    A
E4C5 CA060B             JP    Z,ERRIFC
E4C8 23                 INC   HL
E4C9 5E                 LD    E,(HL)
E4CA 23                 INC   HL
E4CB 56                 LD    D,(HL)
E4CC 1A                 LD    A,(DE)
E4CD 0E05               LD    C,5
E4CF FE20               CP    ' '
E4D1 2815               JR    Z,GETBN1
E4D3 0D                 DEC   C
E4D4 FE4E               CP    'N'
E4D6 2810               JR    Z,GETBN1
E4D8 0D                 DEC   C
E4D9 FE54               CP    'T'
E4DB 280B               JR    Z,GETBN1
E4DD D630               SUB   '0'
E4DF DA060B             JP    C,ERRIFC
E4E2 FE03               CP    3
E4E4 D2060B             JP    NC,ERRIFC
E4E7 4F                 LD    C,A
E4E8 79         GETBN1:LD     A,C
E4E9 E1                 POP   HL
E4EA C9                 RET                         ;ACC , C  are MODE


                ;-------- POKE --------
                ;from ED9F ,ADDRESS IS PUSHED
                ;
E4EB            POKE:
E4EB F1                 POP   AF
E4EC CD77E5             CALL  WED9F
E4EF CF                 RST   8H
E4F0 2C                 DB    ','
E4F1 3E05               LD    A,5
E4F3 F5                 PUSH  AF                    ;SAVE MODE
E4F4 CDA318             CALL  GTBYTE
E4F7 F5                 PUSH  AF                    ;DATA
E4F8 2B                 DEC   HL
E4F9 D7                 RST   10H
E4FA 2809               JR    Z,POKE3
                ;      --- BANK NAME ---
E4FC CF                 RST   8
E4FD 2C                 DB    ','
E4FE CDBCE4             CALL  GETBN
E501 C1                 POP   BC                    ;DATA
E502 D1                 POP   DE                    ;POP OLD MODE
E503 F5                 PUSH  AF                    ;SAVE NEW MODE
E504 C5                 PUSH  BC                    ;DATA

E505            POKE3:
E505 C1                 POP   BC                    ;B=DATA
```

```
E506 F1                 POP  AF
E507 3273E5             LD   (MODE),A
E50A D1                 POP  DE
E50B ED5371E5           LD   (ADRS),DE
E50F E5                 PUSH HL                 ;TEXT POINTER
E510 2A71E5             LD   HL,(ADRS)
E513 CD28E5             CALL IFSWAP
E516 CD99E4             CALL SETMDE
E519 70                 LD   (HL),B
E51A D35F               OUT  (5FH),A
E51C 3EFF               LD   A,255
E51E D371               OUT  (71H),A
E520 3AC2E6             LD   A,(PORT31)
E523 D331               OUT  (31H),A
E525 FB                 EI
E526 E1                 POP  HL
E527 C9                 RET
               ;
               ;  ---- SWAP? ----
               ;
E528           IFSWAP:
E528 3A73E5             LD   A,(MODE)
E52B FE03               CP   3
E52D D0                 RET  NC                 ;MODE <> G-RAM
E52E 1100C0             LD   DE,0C000H
E531 E7                 RST  20H                ;CP HL,DE
E532 D8                 RET  C                  ;HL < C000H

E533 F3                 DI
E534 78                 LD   A,B                ;DATA
E535 215084             LD   HL,SWAREA
E538 117AE5             LD   DE,WKBACK
E53B 010600             LD   BC,6
E53E EDB0               LDIR

E540 E1                 POP  HL                 ;RET ADRS
E541 F5                 PUSH AF                 ;DATA
E542 3ED3               LD   A,0D3H             ;OUT
E544 115084             LD   DE,SWAREA
E547 12                 LD   (DE),A
E548 3A73E5             LD   A,(MODE)
E54B C65C               ADD  A,5CH
E54D 13                 INC  DE
E54E 12                 LD   (DE),A
E54F 23                 INC  HL
E550 23                 INC  HL
E551 23                 INC  HL
E552 13                 INC  DE
E553 010300             LD   BC,3
E556 EDB0               LDIR
E558 3EC9               LD   A,0C9H
E55A 12                 LD   (DE),A
E55B C1                 POP  BC
E55C E5                 PUSH HL                 ;NEW RETURN ADRS
E55D 2A71E5             LD   HL,(ADRS)

E560 CD5084             CALL SWAREA

E563 C5                 PUSH BC
E564 217AE5             LD   HL,WKBACK
E567 115084             LD   DE,SWAREA
E56A 010600             LD   BC,6
E56D EDB0               LDIR
E56F C1                 POP  BC

E570 C9                 RET

               ;-------- WORK AREA --------
               ;
E571 0000      ADRS:    DW   0
E573 00        MODE:    DB   0
E574 000000    WED27:   DB   0,0,0
```

```
E577 000000   WED9F: DB    0,0,0
E57A 00000000 WKBACK:DW    0,0,0
E57E 0000

E580                 END
```

付1-2 N88-BASIC復活

```
                ;-------------------------
                ;   PC-TecKnow 8800mkII
                ;  << RECOVER PROGRAM >>
                ;-------------------------
                ;        relocatable

                      ORG   0F260H

05BD            SETLNK:EQU  05BDH          ; SET ALL LINK POINTERS
1BBB            ADRLIN:EQU  1BBBH          ; CHANGE ADRS TO LINE#
44D5            WNTORL:EQU  44D5H          ; WINDOW TO REAL ADRS

E658            TXTBGN:EQU  0E658H         ; TEXT SATRT ADRS
EB18            TXTEND:EQU  0EB18H         ; TEXT END ADRS+1
EB1A            LABELF:EQU  0EB1AH         ; SHOW THAT LABELS ARE REGISTERD


F260            RECOVR:
F260 2A58E6           LD    HL,(TXTBGN)
F263 3601             LD    (HL),1

F265 CDBD05           CALL  SETLNK
F268 CDBB1B           CALL  ADRLIN
F26B CDD544           CALL  WNTORL
F26E 23               INC   HL
F26F 2218EB           LD    (TXTEND),HL

F272 AF               XOR   A
F273 321AEB           LD    (LABELF),A
F276 FF               DB    0FFH           ; RETURN TO MONITOR

F277                  END
```

付1-3 ROLL文サブルーチン

```
                    ;-----------------------
                    ;  PC-TechKnow 8800mkII
                    ;  << SCROLL COMMAND >>
                    ;-----------------------
                    ;
                    ;  DUMMY = USR(SCROLL.BYTE)
                    ;
                    ;        relocatable

                           ORG  0BF80H

005C                GVRAM0:EQU  5CH              ; OUTPUT PORT FOR SELECT GVRAM0
005D                GVRAM1:EQU  5DH              ; OUTPUT PORT FOR SELECT GVRAM1
005F                MAINRM:EQU  5FH              ; OUTPUT PORT FOR SELECT MAIN RAM

C000                GVRTOP:EQU  0C000H           ; GRAPHIC VRAM TOP ADDRESS
3E80                GVRBYT:EQU  16000
FE80                GVREND:EQU  GVRTOP+GVRBYT


BF80                START:
                    ;----- GET PARAMETER
BF80 7E                    LD   A,(HL)
BF81 23                    INC  HL
BF82 66                    LD   H,(HL)
BF83 6F                    LD   L,A               ; HL = SCROLL.BYTES

                    ;-----  SET PARAMETER
BF84 E5                    PUSH HL
BF85 E5                    PUSH HL
BF86 1100C0                LD   DE,GVRTOP
BF89 19                    ADD  HL,DE
BF8A E3                    EX   (SP),HL
BF8B E5                    PUSH HL
BF8C 21803E                LD   HL,GVRBYT
BF8F C1                    POP  BC
BF90 B7                    OR   A
BF91 ED42                  SBC  HL,BC
BF93 4D                    LD   C,L
BF94 44                    LD   B,H
BF95 E1                    POP  HL

                    ;----- ROLL GVRAM0
BF96 E5                    PUSH HL
BF97 D5                    PUSH DE
BF98 C5                    PUSH BC

BF99 F3                    DI
BF9A D35C                  OUT  (GVRAM0),A
BF9C EDB0                  LDIR
BF9E D35F                  OUT  (MAINRM),A
BFA0 FB                    EI

                    ;----- MOVE GVRAM1 TO GVRAM0
BFA1 E1                    POP  HL
BFA2 C1                    POP  BC

BFA3 C5                    PUSH BC
BFA4 E5                    PUSH HL

BFA5 1180FE                LD   DE,GVREND
BFA8 7C                    LD   A,H
BFA9 F6C0                  OR   0C0H
BFAB 67                    LD   H,A
BFAC                MOVE:
BFAC F3                    DI
BFAD D35D                  OUT  (GVRAM1),A
BFAF 0A                    LD   A,(BC)
```

```
BFB0 D35C            OUT   (GVRAM0),A
BFB2 77              LD    (HL),A
BFB3 D35F            OUT   (MAINRM),A
BFB5 FB              EI

BFB6 23              INC   HL
BFB7 03              INC   BC
BFB8 E7              RST   20H                ; CP HL,DE
BFB9 20F1            JR    NZ,MOVE

               ;----- ROLL GVRAM1
BFBB C1              POP   BC
BFBC D1              POP   DE
BFBD E1              POP   HL

BFBE C5              PUSH BC

BFBF F3              DI
BFC0 D35D            OUT   (GVRAM1),A
BFC2 EDB0            LDIR
BFC4 D35F            OUT   (MAINRM),A
BFC6 FB              EI

               ;----- ERASE NEW LINES ON GVRAM1
BFC7 E1              POP   HL

BFC8 7C              LD    A,H
BFC9 F6C0            OR    0C0H
BFCB 67              LD    H,A

BFCC 5D              LD    E,L
BFCD 54              LD    D,H
BFCE 13              INC   DE
BFCF C1              POP   BC

BFD0 F3              DI
BFD1 D35D            OUT   (GVRAM1),A
BFD3 3600            LD    (HL),0
BFD5 EDB0            LDIR
BFD7 D35F            OUT   (MAINRM),A
BFD9 FB              EI

BFDA C9              RET

BFDB                 END
```

付1-4 アトリビュートセットサブルーチン

```
               ;------------------------
               ;  PC-TechKnow 8800mkII
               ;  << ATTRIBUTE SET >>
               ;------------------------
               ;
               ;  DUMMY = USR(VRAM.ADRS)
               ;
               ;       relocatable

                     ORG   0F320H

4351           PUTATR:EQU  4351H

F320           ATRSET:
F320 0E00            LD    C,0                ; C <- attribute code
F322 7E              LD    A,(HL)
F323 23              INC   HL
F324 66              LD    H,(HL)
F325 6F              LD    L,A                ; HL <- adrs to set in VRAM
F326 CD5143          CALL  PUTATR
F329 C9              RET

F32A                 END
```

付1-5 グラフィックデータ書き込みサブルーチン

```
                  ;-----------------------
                  ;  PC-TechKnow 8800mkII
                  ;  <<  GRAPHIC SUB  >>
                  ;-----------------------
                  ;        relocateble

                         ORG   866AH

F260              DATAAD:EQU   0F260H            ; POINTER TO GRAPHIC DATA


866A 00                  DB    0                 ; TO STOP HELP MSG OF MONITOR
866B              START:
866B 7E                  LD    A,(HL)
866C 23                  INC   HL
866D 66                  LD    H,(HL)
866E 6F                  LD    L,A               ; HL = GVRAM.ADRS

866F ED5B60F2            LD    DE,(DATAAD)

8673 EB                  EX    DE,HL
8674 4E                  LD    C,(HL)
8675 23                  INC   HL
8676 46                  LD    B,(HL)
8677 23                  INC   HL

8678 EB                  EX    DE,HL             ; DE = DATA.POINTER
8679              GS0:
8679 C5                  PUSH BC                 ; BC = DATA.SIZE
867A              GS1:
867A F3                  DI
867B 1A                  LD    A,(DE)
867C D35C                OUT   (5CH),A
867E 77                  LD    (HL),A
867F D35F                OUT   (5FH),A
8681 13                  INC   DE
8682 1A                  LD    A,(DE)
8683 D35D                OUT   (5DH),A
8685 77                  LD    (HL),A
8686 D35F                OUT   (5FH),A
8688 13                  INC   DE
8689 1A                  LD    A,(DE)
868A D35E                OUT   (5EH),A
868C 77                  LD    (HL),A
868D D35F                OUT   (5FH),A
868F 13                  INC   DE
8690 FB                  EI

8691 23                  INC   HL
8692 10E6                DJNZ GS1

8694 C1                  POP   BC
8695 C5                  PUSH BC
8696 3E50                LD    A,80
8698 90                  SUB   B
8699 4F                  LD    C,A
869A 0600                LD    B,0
869C 09                  ADD   HL,BC             ; HL=HL+(80-B)
869D C1                  POP   BC
869E 0D                  DEC   C
869F 20D8                JR    NZ,GS0
86A1 C9                  RET

86A2                     END
```

付1-6 Print Lprint

```
                ;----------------------
                ; PC-TechKnow 8800mkII
                ;  << PRINT/LPRINT >>
                ;----------------------
                ;
                ;    POLL ON/OFF
                ;
                ;       relocatable

                        ORG   0F260H

0095            ON:     EQU   095H
00EE            OFF:    EQU   0EEH

0393            ERRSYN:EQU    0393H

E64C            LPTFLG:EQU    0E64CH

                ;----- POLL COMMAND ( from 0EEB6H )
                ;
F260            POLL:
F260 FE95               CP    ON
F262 280C               JR    Z,POLL1

F264 FEEE               CP    OFF
F266 C29303             JP    NZ,ERRSYN

F269 D7                 RST   10H
F26A C29303             JP    NZ,ERRSYN
F26D AF                 XOR   A
F26E 1806               JR    POLL2
F270            POLL1:
F270 D7                 RST   10H
F271 C29303             JP    NZ,ERRSYN
F274 3E01               LD    A,1
F276            POLL2:
F276 3286F2             LD    (PLPFLG),A
F279 C9                 RET

                ;----- TOP OF PRINT ( from 0EDCFH )
                ;
F27A            PRNTOP:
F27A F5                 PUSH  AF
F27B 3A86F2             LD    A,(PLPFLG)
F27E B7                 OR    A
F27F 2803               JR    Z,NOTPLP

F281 324CE6             LD    (LPTFLG),A
F284            NOTPLP:
F284 F1                 POP   AF
F285 C9                 RET

                ;----- WORKING AREA
                ;
F286 00         PLPFLG:DB     0
                ;
F287                    END
```

付1-7 テキスト画面のコピー

```
               ;------------------------
               ;  PC-TechKnow 8800mkII
               ;     << TEXT COPY >>
               ;------------------------
               ;
               ; PTYPE has the type of printer
               ;         06: PC-8023(C)
               ;         0E: PC-8821/8822,NM-9100/9200,NK-3618
               ;         16: PC-PR201
               ;         1E: NM-9300/9400
               ;

                       ORG   08000H

35C2           BRKCHK:EQU   35C2H             ; BREAK CHECK (CY=1 if STOP)
429D           CSRVRM:EQU   429DH             ; CURSOR POS -> VRAM ADRS
4452           GETVRM:EQU   4452H             ; GET CHAR & ATTR AT HL

E6C1           PORT40:EQU   0E6C1H            ; BACK UP OF PORT 40H
EF88           YWIDTH:EQU   0EF88H            ; Y OF WIDTH
EF89           XWIDTH:EQU   0EF89H            ; X OF WIDTH

8000 0E        PTYPE: DB    0EH               ; DEFAULT IS PC-8821
8001 00               DB    00
8002 C31580           JP    START
               ;
               ;====     WORK AREA     ====

8005 00000000 BUFFER:DW    0,0,0,0
8009 00000000
800D 00000000 BUF2:  DW    0,0,0,0
8011 00000000

               ;====      CODES        ====
               ;
8015           START: EQU   $

8015 D3EB             OUT   (0EBH), A         ; DESELECT KANJI ROM
8017 CD6980           CALL LPTMDE

801A 2E01             LD    L, 1              ; SET Y-POS TO 0
801C           TCOPY1:
801C CDC235           CALL BRKCHK
801F 3825             JR    C, TCOPY3
8021 E5               PUSH HL
8022 CDE780           CALL TOPLIN
8025 E1               POP   HL
8026 2601             LD    H, 1              ; SET X-POS TO 0
8028           TCOPY2:
8028 E5               PUSH HL
8029 CD9D42           CALL CSRVRM             ; (X,Y) -> VRAM ADRS
802C CD5244           CALL GETVRM             ; (ADRS) -> B:char, C:attr
802F CD4A81           CALL PRINTC             ; LPRINT ONE CHAR
8032 E1               POP   HL
8033 24               INC   H                 ; ++X
8034 3A89EF           LD    A, (XWIDTH)       ; 40 or 80
8037 BC               CP    H
8038 30EE             JR    NC, TCOPY2

803A E5               PUSH HL
803B CD3F81           CALL ENDLIN
803E E1               POP   HL
803F 2C               INC   L
8040 3A88EF           LD    A, (YWIDTH)
8043 BD               CP    L
8044 30D6             JR    NC, TCOPY1

8046           TCOPY3:
```

```
8046 064E               LD   B, 'N'
8048 3A0080             LD   A, (PTYPE)
804B FE06               CP   06H                 ; PC-8023?
804D 2802               JR   Z, TCOPY4           ; YES
804F 0648               LD   B, 'H'
8051          TCOPY4:
8051 3E1B               LD   A, 27
8053 CD2D82             CALL LPTBYT
8056 78                 LD   A, B
8057 CD2D82             CALL LPTBYT
805A 216180             LD   HL, ENDMDE
805D CD2482             CALL LPTSTR
8060 C9                 RET

8061 0F1B411B ENDMDE:DB    15, 27,'A',27,']',13,10,0
8065 5D0D0A00

              ;==== PRINTER MODE SET
8069          LPTMDE:
8069 060F               LD   B, 15                     ; SO CODE
806B 3A89EF             LD   A, (XWIDTH)
806E FE32               CP   50
8070 3002               JR   NC, LPTMD0
8072 060E               LD   B, 14                     ; SI CODE
8074          LPTMD0:
8074 78                 LD   A, B
8075 CD2D82             CALL LPTBYT
8078 3A0080             LD   A, (PTYPE)
807B D606               SUB  6
807D 6F                 LD   L, A
807E 0F                 RRCA
807F 85                 ADD  A, L
8080 6F                 LD   L, A
8081 2600               LD   H, 0
8083 11B780             LD   DE, MODSEQ
8086 19                 ADD  HL, DE
8087 5E                 LD   E, (HL)
8088 3A88EF             LD   A, (YWIDTH)
808B FE15               CP   21
808D 3003               JR   NC, LPTMD1
808F 23                 INC  HL
8090 5E                 LD   E, (HL)
8091 2B                 DEC  HL
8092          LPTMD1:
8092 D5                 PUSH DE
8093 23                 INC  HL
8094 23                 INC  HL
8095 CD2482             CALL LPTSTR
8098 D1                 POP  DE
8099 1600               LD   D, 0
809B 21A380             LD   HL, LFWID
809E 19                 ADD  HL, DE
809F CD2482             CALL LPTSTR
80A2 C9                 RET


80A3 1B543132 LFWID: DB    27,'T12',0           ; 0
80A7 00
80A8 1B543135        DB    27,'T15',0           ; 5
80AC 00
80AD 1B543136        DB    27,'T16',0           ; 10
80B1 00
80B2 1B543230        DB    27,'T20',0           ; 15
80B6 00

80B7          MODSEQ:
80B7 0A0F1B51        DB    10,15,27,'Q',27,'[',0,0,0,0,0,0          ; PC-8023
80BB 1B5B0000
80BF 00000000
80C3 0A0F1B48        DB    10,15,27,'H',27,'>',0,0,0,0,0,0          ; PC-8821
80C7 1B3E0000
80CB 00000000
```

```
80CF 00051B48         DB    00,05,27,'H',0,0,0,0,0,0,0,0          ; PR-201
80D3 00000000
80D7 00000000
80DB 0A0F1B51         DB    10,15,27,'Q',26,'A',26,'F',27,'>',0,0  ; NM-9300/9400
80DF 1A411A46
80E3 1B3E0000

               ;==== TOP OF LINE PROCEDURE
               ;
80E7           TOPLIN:
80E7 3A0080           LD    A, (PTYPE)
80EA 6F               LD    L, A
80EB 2600             LD    H, 0
80ED 11F980           LD    DE, TOPS80 - 6
80F0 3A89EF           LD    A, (XWIDTH)
80F3 FE32             CP    50
80F5 3003             JR    NC, TOPLN1
80F7 111981           LD    DE, TOPS40 - 6
80FA           TOPLN1:
80FA 19               ADD   HL, DE
80FB CD2482           CALL  LPTSTR
80FE C9               RET

80FF           TOPS80:
80FF 1B533036         DB    27,'S0640',0,0      ; PC-8023
8103 34300000
8107 1B493036         DB    27,'I0640',0,0      ; PC-8821
810B 34300000
810F 1B493036         DB    27,'I0640',0,0      ; PR-201
8113 34300000
8117 1B4A3036         DB    27,'J0640',0,0      ; NM-9300/9400
811B 34300000
811F           TOPS40:
811F 1B533033         DB    27,'S0320',0,0      ; PC-8023
8123 32300000
8127 1B493033         DB    27,'I0320',0,0      ; PC-8821
812B 32300000
812F 1B493033         DB    27,'I0320',0,0      ; PR-201
8133 32300000
8137 1B4A3033         DB    27,'J0320',0,0      ; NM-9300/9400
813B 32300000


               ;==== END OF LINE PROCEDURE
813F           ENDLIN:
813F 3E0D             LD    A, 13               ; CARRIGE RETURN
8141 CD2D82           CALL  LPTBYT
8144 3E0A             LD    A, 10               ; LINE FEED
8146 CD2D82           CALL  LPTBYT
8149 C9               RET

               ;==== PRINT A SCREEN CHARACTER
814A           PRINTC:
814A C5               PUSH  BC
814B CDC181           CALL  MKFONT              ; PREPARE THE FONT OF THE CHAR
814E C1               POP   BC

814F CB59             BIT   3, C
8151 200C             JR    NZ, PRNC1           ; COLOR ATTRIBUTE
               ;    ---- MONOCHROME ATTRIBUTE
8153 CB51             BIT   2, C
8155 C4AA81           CALL  NZ, REVRSE          ; REVERSE BIT ON
8158 CB41             BIT   0, C
815A C4B681           CALL  NZ, SECRET          ; SECRET BIT ON
815D 1806             JR    PRNC2
               ;    ---- COLOR ATTRIBUTE
815F           PRNC1:
815F 79               LD    A, C
8160 E6E0             AND   0E0H
8162 CCB681           CALL  Z, SECRET           ; COLOR 0

               ;    ---- SEND ONE CHARACTER
```

```
8165            PRNC2:
                ;  E = 0:   8 dots
                ;      1:  16 dots
                ;      3:  24 dots

8165 3A0080             LD   A, (PTYPE)
8168 1E00               LD   E, 0
816A D606               SUB  6                    ; PC-8023?
816C 2807               JR   Z, PRNC3             ; Yes. 8 dots mode
816E 1C                 INC  E
816F D611               SUB  17                   ; PC-8821 or PR-201?
8171 3802               JR   C, PRNC3             ; Yes. 16 dots mode
8173 1E03               LD   E, 3
8175            PRNC3:
8175 210580             LD   HL, BUFFER
8178 1608               LD   D, 8
817A            PRNC3A:
817A 4E                 LD   C, (HL)
817B 3E80               LD   A, 80H
817D 0608               LD   B, 8
817F            PRNC3B:
817F CB09               RRC  C
8181 1F                 RRA
8182 DCA481             CALL C, PRNONE
8185 CB43               BIT  0, E
8187 2814               JR   Z, PRNC3C
8189 CB01               RLC  C
818B CB09               RRC  C
818D 1F                 RRA
818E DCA481             CALL C, PRNONE
8191 CB4B               BIT  1, E
8193 2808               JR   Z, PRNC3C
8195 CB01               RLC  C
8197 CB09               RRC  C
8199 1F                 RRA
819A DCA481             CALL C, PRNONE

819D            PRNC3C:
819D 10E0               DJNZ PRNC3B

819F 23                 INC  HL
81A0 15                 DEC  D
81A1 20D7               JR   NZ, PRNC3A
81A3 C9                 RET

81A4            PRNONE:
81A4 CD2D82             CALL LPTBYT
81A7 3E80               LD   A, 80H
81A9 C9                 RET


                ;==== MODIFY FONT WITH ATTRIBUTE
                ;----REVERSE FONT
81AA            REVRSE:
81AA 210580             LD   HL, BUFFER
81AD 0608               LD   B, 8
81AF            REVR1:
81AF 7E                 LD   A, (HL)
81B0 2F                 CPL
81B1 77                 LD   (HL), A
81B2 23                 INC  HL
81B3 10FA               DJNZ REVR1
81B5 C9                 RET

                ;----SECRET FONT
81B6            SECRET:
81B6 210580             LD   HL, BUFFER
81B9 0608               LD   B, 8
81BB AF                 XOR  A
81BC            SECR1:
81BC 77                 LD   (HL), A
81BD 23                 INC  HL
```

```
81BE 10FC             DJNZ SECR1
81C0 C9               RET

               ;==== PREPARE THE FONT OF THE CHAR
81C1           MKFONT:

81C1 CB59             BIT  3, C
81C3 2006             JR   NZ, MKF1
81C5 CB79             BIT  7, C              ; MONOCHROME MODE
81C7 203E             JR   NZ, GRAPH
81C9 1804             JR   MKF2
81CB           MKF1:
81CB CB61             BIT  4, C              ; COLOR MODE
81CD 2038             JR   NZ, GRAPH
81CF           MKF2:
               ;    ---- CHARACTER
81CF 68               LD   L, B              ; CHAR CODE
81D0 2602             LD   H, 2              ; HL <- (CHAR CODE OR &h200)
81D2 29               ADD  HL, HL
81D3 29               ADD  HL, HL            ; HL <- KANJI ROM ADRS (1/4)

81D4 110D80           LD   DE, BUF2
81D7 0604             LD   B, 4
81D9           MKF3:
81D9 7D               LD   A, L
81DA D3E8             OUT  (0E8H), A         ; SET ADRS LOW
81DC 7C               LD   A, H
81DD D3E9             OUT  (0E9H), A         ; SET ADRS HIGH
81DF D3EA             OUT  (0EAH), A
81E1 00               NOP
81E2 00               NOP
81E3 DBE9             IN   A, (0E9H)         ; READ FIRST BYTE
81E5 12               LD   (DE), A
81E6 13               INC  DE
81E7 DBE8             IN   A, (0E8H)         ; READ SECOND BYTE
81E9 12               LD   (DE), A
81EA 13               INC  DE
81EB D3EB             OUT  (0EBH), A
81ED 23               INC  HL                ; INCREMENT KANJI ADRS
81EE 10E9             DJNZ MKF3

               ;        ---- ROTETE -90°
81F0 110580           LD   DE, BUFFER
81F3 0E08             LD   C, 8
81F5           MKF4:
81F5 210D80           LD   HL, BUF2
81F8 0608             LD   B, 8
81FA AF               XOR  A
81FB           MKF5:
81FB CB06             RLC  (HL)
81FD 1F               RRA
81FE 23               INC  HL
81FF 10FA             DJNZ MKF5
8201 12               LD   (DE), A
8202 13               INC  DE
8203 0D               DEC  C
8204 20EF             JR   NZ, MKF4

8206 C9               RET


               ;    ---- MAKE LOW RESOLUTION GRPHICS
8207           GRAPH:
8207 210580           LD   HL, BUFFER
820A 1E02             LD   E, 2
820C           GRPH1:
820C AF               XOR  A
820D 0E04             LD   C, 4
820F           GRPH2:
820F CB08             RRC  B
8211 1F               RRA
8212 1F               RRA
```

```
8213 0D                DEC   C
8214 20F9              JR    NZ, GRPH2

8216 4F                LD    C, A
8217 87                ADD   A, A
8218 81                ADD   A, C

8219 0E04              LD    C, 4
821B           GRPH3:
821B 77                LD    (HL), A
821C 23                INC   HL
821D 0D                DEC   C
821E 20FB              JR    NZ, GRPH3

8220 1D                DEC   E
8221 20E9              JR    NZ, GRPH1

8223 C9                RET


               ;==== LPT OUT SUBROUTINES

               ;---- LPT OUT A STRING
8224           LPTSTR:
8224 7E                LD    A, (HL)
8225 B7                OR    A
8226 C8                RET   Z
8227 CD2D82            CALL  LPTBYT
822A 23                INC   HL
822B 18F7              JR    LPTSTR

               ;---- LPT OUT A BYTE
822D           LPTBYT:
822D F5                PUSH  AF
822E           LPTBT1:
822E DB40              IN    A, (40H)
8230 0F                RRCA
8231 38FB              JR    C, LPTBT1
8233 F1                POP   AF
8234 D310              OUT   (10H), A
8236 F5                PUSH  AF
8237 F3                DI
8238 3AC1E6            LD    A, (PORT40)
823B E6FE              AND   0FEH
823D D340              OUT   (40H), A
823F F601              OR    1
8241 D340              OUT   (40H), A
8243 FB                EI
8244 F1                POP   AF
8245 C9                RET

8246                   END
```

付1-8 グラフィック画面のコピー

```
                ;------------------------------
                ;     PC-TechKnow 8800mkII
                ;   << GRAPHIC SCREEN COPY >>
                ;------------------------------
                ;
                ; PTYPE  (PRINTER TYPE)
                ;     0E: PC-8821/8822,NM-9100/9200,NK-3618
                ;     16: PR-201
                ;     1E: NM-9300/9400
                ;
                        ORG   08000H

35C2            BRKCHK:EQU   35C2H              ; BREAK CHECK (CY=1 if STOP)
8300            GPEEK: EQU   8300H              ; GVRAM PEEK ROUTINE
E6C1            PORT40:EQU   0E6C1H             ; BACK UP OF PORT 40H


8000 0E         PTYPE: DB    0EH
8001 00         MASK:  DB    00H
8002 C31880            JP    START
8005 000000            DB    0,0,0              ; TO LOCATE CPATRN AT ORG+10H

                ;====    WORK   AREA    ====
8008 00000000 COLORB:DB    0,0,0,0,0,0,0,0      ; MSB --> LSB
800C 00000000

                ;====    DOT   PATTERN   ====
8010 00202151 CPATRN:DB    0,20H,21H,51H,69H,9EH,0DEH,0FFH
8014 699EDEFF

                ;====       CODES          ====
8018            START:
8018 DB70              IN    A, (70H)           ; READ OAR
801A F5                PUSH AF                  ; SAVE OAR VALUE
801B 3E80              LD    A, 80H
801D D370              OUT   (70H), A           ; RESET WINDOW

801F 212381            LD    HL, GPEEKR
8022 110083            LD    DE, GPEEK
8025 010A00            LD    BC, GPEEKE - GPEEKR
8028 EDB0              LDIR                     ; MOVE GRAHIC PEEK ROUTINE

802A CD7080            CALL  LPTMDE             ; PRINTER MODE SET
802D DD2130FE          LD    IX, 0C000H+15920   ; ADDRESS OF (0,199)
8031 115000            LD    DE, 80             ; # OF BYTES / 640DOTS

8034            GCOPY1:
8034 CDC235            CALL  BRKCHK
8037 3826              JR    C, GCOPY3          ; IF STOP
8039 D5                PUSH  DE
803A CDB180            CALL  TOPLIN             ; SET TO BIT IMAGE MODE
803D 06C8              LD    B, 200
803F            GCOPY2:
803F C5                PUSH  BC
8040 CDC980            CALL  SETCBF             ; SET COLOR BUFFER
8043 CDF180            CALL  CNGCTP             ; COLOR# -> PATTERN
8046 CD0781            CALL  OUTPTN             ; OUT PATTERN TO PRINTER
8049 11B0FF            LD    DE, -80
804C DD19              ADD   IX, DE             ; MOVE TO THE BYTE ABOVE
804E C1                POP   BC
804F 10EE              DJNZ  GCOPY2

8051 CDBF80            CALL  ENDLIN             ; CARRIGE RETURN
8054 11813E            LD    DE, 16001
8057 DD19              ADD   IX, DE
8059 D1                POP   DE
805A 1B                DEC   DE
```

```
805B 7A                 LD   A, D
805C B3                 OR   E
805D 20D5               JR   NZ, GCOPY1

805F          GCOPY3:
805F F1                 POP  AF
8060 D370               OUT  (70H), A          ; RESTORE OAR
8062 216980             LD   HL, ENDMDE
8065 CD2D81             CALL LPTSTR            ; NORMALIZE PRINTER MODE
8068 C9                 RET

8069 1B411B5D ENDMDE:DB   27,'A',27,')',13,10,0
806D 0D0A00

              ;---- SET PRINTER MODE
8070          LPTMDE:
8070 3A0080             LD   A, (PTYPE)        ; PRINTER TYPE
8073 D60E               SUB  0EH
8075 6F                 LD   L, A
8076 0F                 RRCA
8077 85                 ADD  A, L              ; A <- A x 1.5
8078 6F                 LD   L, A
8079 2600               LD   H, 0
807B 118D80             LD   DE, MDESEQ
807E 19                 ADD  HL, DE
807F CD2D81             CALL LPTSTR
8082 3E1B               LD   A, 27
8084 CD3681             CALL LPTBYT
8087 3E48               LD   A, 'H'
8089 CD3681             CALL LPTBYT
808C C9                 RET

808D          MDESEQ:
808D 1B543136           DB   27,'T16',27,'>',0,0,0,0,0,0          ; PC-8821
8091 1B3E0000
8095 00000000
8099 1B543132           DB   27,'T12',0,0,0,0,0,0,0,0             ; PR-201
809D 00000000
80A1 00000000
80A5 1B3E1A41           DB   27,'>',26,'A',26,'F',27,'T16',0,0    ; NM-9300
80A9 1A461B54
80AD 31360000


              ;---- TOP OF LINE PROCEDURE
80B1          TOPLIN:
80B1 21B880             LD   HL, TOPSEQ
80B4 CD2D81             CALL LPTSTR
80B7 C9                 RET

80B8 1B493038 TOPSEQ:DB   27,'I0800',0
80BC 303000

              ;---- END OF LINE PROCEDURE
80BF          ENDLIN:
80BF 21C680             LD   HL, ENDSEQ
80C2 CD2D81             CALL LPTSTR
80C5 C9                 RET

80C6 0D0A00   ENDSEQ:DB   13, 10, 0

              ;==== SET COLOR BUFFER
80C9          SETCBF:
80C9 0E5E               LD   C, 5EH            ; AT FIRST, READ GREEN
80CB 1603               LD   D, 3              ; READ 3 PLANES
80CD          STCBF1:
80CD CD0083             CALL GPEEK             ; A <- GVRAM(C,IX)
80D0 210880             LD   HL, COLORB
80D3 0608               LD   B, 8
80D5          STCBF2:
80D5 07                 RLCA
```

```
80D6 CB16              RL   (HL)
80D8 23                INC  HL
80D9 10FA              DJNZ STCBF2

80DB 0D                DEC  C                   ; CHANGE COLOR PLANE
80DC 15                DEC  D
80DD 20EE              JR   NZ, STCBF1

80DF 210880            LD   HL, COLORB
80E2 3A0180            LD   A, (MASK)
80E5 4F                LD   C, A
80E6 0608              LD   B, 8
80E8            STCBF3:
80E8 7E                LD   A, (HL)
80E9 A9                XOR  C
80EA E607              AND  7
80EC 77                LD   (HL), A
80ED 23                INC  HL
80EE 10F8              DJNZ STCBF3
80F0 C9                RET

                ;==== CHANGE COLOR TO PATTERN
80F1            CNGCTP:
80F1 010880            LD   BC, COLORB
80F4 111080            LD   DE, CPATRN
80F7 3E08              LD   A, 8
80F9            CGCTP1:
80F9 08                EX   AF, AF'
80FA 0A                LD   A, (BC)
80FB 6F                LD   L, A
80FC 2600              LD   H, 0
80FE 19                ADD  HL, DE
80FF 7E                LD   A, (HL)
8100 02                LD   (BC), A
8101 03                INC  BC
8102 08                EX   AF, AF'
8103 3D                DEC  A
8104 20F3              JR   NZ, CGCTP1
8106 C9                RET

                ;==== OUTPUT PATTERNS
8107            OUTPTN:
8107 0E04              LD   C, 4
8109            OTPTN1:
8109 210880            LD   HL, COLORB
810C 1E02              LD   E, 2
810E            OTPTN2:
810E 0604              LD   B, 4
8110            OTPTN3:
8110 CB06              RLC  (HL)
8112 1F                RRA
8113 CB06              RLC  (HL)
8115 1F                RRA
8116 23                INC  HL
8117 10F7              DJNZ OTPTN3
8119 CD3681            CALL LPTBYT

811C 1D                DEC  E
811D 20EF              JR   NZ, OTPTN2

811F 0D                DEC  C
8120 20E7              JR   NZ, OTPTN1
8122 C9                RET

                ;==== GVRAM PEEK ROUTINE
8123            GPEEKR:
8123 F3                DI
8124 ED79              OUT  (C), A
8126 DD7E00            LD   A, (IX)
8129 D35F              OUT  (5FH), A
812B FB                EI
812C C9                RET
```

```
812D            GPEEKE:

                ;==== LPT OUT SUBROUTINES

                ;---- LPT OUT A STRING
812D            LPTSTR:
812D 7E                 LD    A, (HL)
812E B7                 OR    A
812F C8                 RET   Z
8130 CD3681             CALL  LPTBYT
8133 23                 INC   HL
8134 18F7               JR    LPTSTR

                ;---- LPT OUT A BYTE
8136            LPTBYT:
8136 F5                 PUSH  AF
8137            LPTBT1:
8137 DB40               IN    A, (40H)
8139 0F                 RRCA
813A 38FB               JR    C, LPTBT1
813C F1                 POP   AF
813D D310               OUT   (10H), A
813F F5                 PUSH  AF
8140 F3                 DI
8141 3AC1E6             LD    A, (PORT40)
8144 E6FE               AND   0FEH
8146 D340               OUT   (40H), A
8148 F601               OR    1
814A D340               OUT   (40H), A
814C FB                 EI
814D F1                 POP   AF
814E C9                 RET

814F                    END
```

付1-9 N-BASICで1,200bpsを

```
                ;-------------------------------
                ;     PC-TechKNOW 8800mkII
                ;  << 1200 bps FOR N-BASIC >>
                ;-------------------------------

                        ORG    0F2C5H

0BFD            INITRD:EQU     0BFDH
0C50            INITWR:EQU     0C50H
3BDF            SYNERR:EQU     3BDFH

EA66            COPY:  EQU     0EA66H
F1B6            HOOKRD:EQU     0F1B6H
F1B9            HOOKWR:EQU     0F1B9H

F2C5            CMD:                                ; FROM CMD'S HOOK
F2C5 7E                 LD     A,(HL)
F2C6 D642               SUB    'B'
F2C8 200A               JR     NZ, NOTB
F2CA 3EC9               LD     A, 0C9H              ; CODE OF 'RET'
F2CC            SETOP:
F2CC 32B6F1             LD     (HOOKRD),A
F2CF 32B9F1             LD     (HOOKWR),A
F2D2 D7                 RST    10H
F2D3 C9                 RET

F2D4            NOTB:
F2D4 3C                 INC    A
F2D5 C2DF3B             JP     NZ,SYNERR
F2D8 E5                 PUSH   HL
F2D9 21EAF2             LD     HL,READST
F2DC 22B7F1             LD     (HOOKRD+1),HL
F2DF 21F5F2             LD     HL,WRITST
F2E2 22BAF1             LD     (HOOKWR+1),HL
F2E5 3EC3               LD     A,0C3H               ; CODE OF 'JP'
F2E7 E1                 POP    HL
F2E8 18E2               JR     SETOP

F2EA            READST:
F2EA F1                 POP    AF
F2EB 3A66EA             LD     A,(COPY)
F2EE E60F               AND    0FH
F2F0 F618               OR     18H                  ; 1200 bps MODE
F2F2 C3FD0B             JP     INITRD

F2F5            WRITST:
F2F5 F1                 POP    AF
F2F6 3A66EA             LD     A,(COPY)
F2F9 E60F               AND    0FH
F2FB F61C               OR     1CH                  ; 1200 bps MODE
F2FD C3500C             JP     INITWR

F300                    END
```

付1-10 高速画面クリア

```
                  ;--------------------------
                  ;   PC-TechKnow 8800mkII
                  ;  << HIGH SPEED CLS 2 >>
                  ;--------------------------
                  ;        relocateble

                          ORG   0BF00H

BF00              HSCLS:
BF00 210000               LD    HL, 0
BF03 39                   ADD   HL, SP          ; SAVE SP INTO HL
BF04 110000               LD    DE, 0
BF07 0E5C                 LD    C, 5CH
BF09 F3                   DI
BF0A              HSCLS0:
BF0A 3180FE               LD    SP, 0FE80H      ; END OF GVRAM ADRS
BF0D 3E02                 LD    A, 2
BF0F ED79                 OUT   (C), A          ; SELECT GVRAM
BF11              HSCLS1:
BF11 06FA                 LD    B, 0FAH
BF13              HSCLS2:
BF13 D5                   PUSH DE               ; CLEAR 32bytes
BF14 D5                   PUSH DE
BF15 D5                   PUSH DE
BF16 D5                   PUSH DE
BF17 D5                   PUSH DE
BF18 D5                   PUSH DE
BF19 D5                   PUSH DE
BF1A D5                   PUSH DE
BF1B D5                   PUSH DE
BF1C D5                   PUSH DE
BF1D D5                   PUSH DE
BF1E D5                   PUSH DE
BF1F D5                   PUSH DE
BF20 D5                   PUSH DE
BF21 D5                   PUSH DE
BF22 D5                   PUSH DE
BF23 10EE                 DJNZ HSCLS2
BF25 3D                   DEC   A
BF26 20E9                 JR    NZ, HSCLS1
BF28 0C                   INC   C               ; NEXT GVRAM
BF29 79                   LD    A, C
BF2A FE5F                 CP    5FH
BF2C 38DC                 JR    C, HSCLS0

BF2E D35F                 OUT   (5FH), A        ; SELECT MAIN RAM
BF30 F9                   LD    SP, HL          ; RESTORE SP
BF31 FB                   EI
BF32 C9                   RET
```

付1-11 漢字フォント読み出しサブルーチン

```
                   ;-------------------------
                   ;    PC-TechKnow 8800mkII
                   ;  << READ KANJI DATA >>
                   ;-------------------------
                   ;         relocatable

                          ORG   0F260H

7261               KANJI2:EQU   7261H

F260               RKFSUB:
F260 7E                   LD    A, (HL)
F261 23                   INC   HL
F262 66                   LD    H, (HL)
F263 6F                   LD    L, A
F264 EB                   EX    DE, HL              ; DE <- JIS CODE
F265 F3                   DI
F266 3EFE                 LD    A, 0FEH
F268 D371                 OUT   (71H), A            ; SELECT ROM3
F26A CD6172               CALL  KANJI2              ; READ KANJI DATA
F26D 3EFF                 LD    A, 0FFH
F26F D371                 OUT   (71H), A            ; SELECT N88-ROM
F271 FB                   EI
F272 C9                   RET                       ; RETURN TO BASIC
```

付1-12 高速漢字PUT文

```
                   ;--------------------------
                   ;     PC-TechKnow 8800mkII
                   ;  << KANJI PUT ROUTINE >>
                   ;--------------------------
                   ;
                   ;  POLL "STRING"
                   ;

                          ORG   0866AH

11D3               EVAL:  EQU   11D3H               ; EVALUATE FORMULA
56C9               CSFRFA:EQU   56C9H               ; CHECK AS IF STRING & FREE IT

F027               SLPX:  EQU   0F027H              ; X OF LP (SCREEN COOD)
F029               SLPY:  EQU   0F029H              ; Y OF LP (SCREEN COOD)
F01E               FOREC: EQU   0F01EH              ; FORE GROUND COLOR


866A               POLL:
866A 00                   NOP                       ; TO TERMINATE MESSAGES
866B CDD311               CALL  EVAL
866E E5                   PUSH  HL                  ; TEXT POINTER
866F CDC956               CALL  CSFRFA
8672 7E                   LD    A, (HL)
8673 F5                   PUSH  AF                  ; LENGTH
8674 23                   INC   HL
8675 5E                   LD    E,(HL)
8676 23                   INC   HL
8677 56                   LD    D,(HL)              ; DE=STRING ADRS
8678 D5                   PUSH  DE
8679 D3EB                 OUT   (0EBH),A            ; OFF KANJI ROM
                   ;
                   ;--- CALC Y*80+X+C000H
867B CDEB86               CALL  CALADR
867E E5                   PUSH  HL
867F DDE1                 POP   IX
                   ;
8681               KPNXT:
8681 E1                   POP   HL                  ; STRING ADRS
```

```
8682 F1                 POP   AF
8683 FE02               CP    2
8685 3810               JR    C, KPEND

8687 3D                 DEC   A
8688 3D                 DEC   A
8689 F5                 PUSH  AF
868A 7E                 LD    A,(HL)
868B E67F               AND   7FH
868D 57                 LD    D,A
868E 23                 INC   HL
868F 7E                 LD    A,(HL)
8690 E67F               AND   7FH
8692 5F                 LD    E,A
8693 23                 INC   HL
8694 E5                 PUSH  HL                ; STRING ADRS
8695 1802               JR    KPUT1

8697            KPEND:
8697 E1                 POP   HL                ; TEXT POINTER
8698 C9                 RET
                ;
8699            KPUT1:
8699 EB                 EX    DE,HL
869A 7C                 LD    A,H               ; HIGH BYTE
869B FE30               CP    30H               ; KANJI OR CODE?
869D 300C               JR    NC,KPUT2          ; KANJI
                ;--- CONV. CODE to ROM ADRS
869F 87                 ADD   A,A               ; H+H
86A0 E60E               AND   0EH
86A2 57                 LD    D,A
86A3 7D                 LD    A,L
86A4 0F                 RRCA
86A5 E630               AND   30H
86A7 B2                 OR    D
86A8 57                 LD    D,A
86A9 180A               JR    KPUT3
                ;--- CONV. KANJI to ROM ADRS
86AB            KPUT2:
86AB E61F               AND   1FH
86AD 87                 ADD   A,A
86AE 57                 LD    D,A
86AF 7D                 LD    A,L
86B0 E660               AND   60H
86B2 87                 ADD   A,A
86B3 B2                 OR    D
86B4 57                 LD    D,A
                ;
86B5            KPUT3:
86B5 7D                 LD    A,L
86B6 87                 ADD   A,A
86B7 87                 ADD   A,A
86B8 87                 ADD   A,A
86B9 87                 ADD   A,A
86BA 5F                 LD    E,A
86BB 7A                 LD    A,D
86BC CE00               ADC   A,0
86BE 57                 LD    D,A
                ;
                ;--- PUT ON VRAM
                ;
86BF 0610               LD    B,16
86C1 DDE5               PUSH  IX
86C3 E1                 POP   HL
86C4            KPUT5:
86C4 C5                 PUSH  BC
                ;
86C5 7B                 LD    A,E
86C6 D3E8               OUT   (0E8H),A
86C8 7A                 LD    A,D
86C9 D3E9               OUT   (0E9H),A
86CB 13                 INC   DE
```

```
                   ;
86CC D3EA                  OUT   (0EAH),A              ; START READING
86CE 00                    NOP
86CF 00                    NOP
86D0 DBE9                  IN    A,(0E9H)
86D2 CD0E87                CALL  PUTDAT
86D5 23                    INC   HL
86D6 DBE8                  IN    A,(0E8H)
86D8 CD0E87                CALL  PUTDAT
86DB D3EB                  OUT   (0EBH),A              ; END READING
86DD 014F00                LD    BC,79
86E0 09                    ADD   HL,BC
                   ;
86E1 C1                    POP   BC
86E2 10E0                  DJNZ  KPUT5
                   ;
                   ;--- TO NEXT CHARACTER
86E4 DD23                  INC   IX
86E6 DD23                  INC   IX
86E8 C38186                JP    KPNXT
                   ;
                   ;--- CALCULATE ADDRESS   (LPYx80+LPX¥8+C000H)
86EB               CALADR:
86EB 2A29F0                LD    HL,(SLPY)
86EE 29                    ADD   HL,HL                 ; x2
86EF 29                    ADD   HL,HL                 ; x4
86F0 29                    ADD   HL,HL                 ; x8
86F1 44                    LD    B, H
86F2 4D                    LD    C, L
86F3 29                    ADD   HL,HL                 ; x16
86F4 29                    ADD   HL,HL                 ; x32
86F5 09                    ADD   HL,BC                 ; x40
86F6 29                    ADD   HL,HL                 ; x80
86F7 44                    LD    B, H
86F8 4D                    LD    C, L

86F9 2A27F0                LD    HL,  (SLPX)
86FC CB3C                  SRL   H
86FE CB1D                  RR    L                     ; /2
8700 CB3C                  SRL   H
8702 CB1D                  RR    L                     ; /4
8704 CB3C                  SRL   H
8706 CB1D                  RR    L                     ; /8

8708 09                    ADD   HL,  BC               ; Yx80 + X¥8
8709 0100C0                LD    BC,0C000H
870C 09                    ADD   HL,BC                 ; HL = Yx80 + X¥8 + C000H
870D C9                    RET

                   ;---- PUT A BYTE ON GVRAM
870E               PUTDAT:
870E D5                    PUSH  DE
870F 57                    LD    D,  A
8710 0E5C                  LD    C,  5CH
8712 3A1EF0                LD    A,  (FOREC)
8715 0603                  LD    B,  3
8717 F3                    DI
8718               PUTDT1:
8718 ED79                  OUT   (C),  A
871A 0F                    RRCA
871B 3003                  JR    NC, PUTDT2
871D 72                    LD    (HL),  D
871E 1802                  JR    PUTDT3
8720               PUTDT2:
8720 3600                  LD    (HL),  0
8722               PUTDT3:
8722 0C                    INC   C
8723 10F3                  DJNZ  PUTDT1
8725 D35F                  OUT   (05FH),  A
8727 FB                    EI
8728 D1                    POP   DE
8729 C9                    RET

872A                       END
```

付1-13 モニタでテキストラムを読む

```
                    ;---------------------------------
                    ;       PC-TechKnow 8800mkI
                    ;  << READ TEXT RAM IN MONITOR >>
                    ;---------------------------------
                            ORG   0F2C0H

E6C2                PORT31:EQU   0E6C2H

                    ;======== MONITOR PATCHING ROUTINE ========
F2C0                MONPAT:                          ; FROM 0E66CH
F2C0 F1                     POP   AF
F2C1 21E3F2                 LD    HL, SELRAM         ; CHANGE HERE TO RELOCATE
F2C4 11A0F1                 LD    DE, 0F1A0H
F2C7 010C00                 LD    BC, 12
F2CA EDB0                   LDIR
F2CC 3E03                   LD    A, 3
F2CE 32E1F1                 LD    (0F1E1H), A        ; BANK SWITCH
F2D1 3E48                   LD    A, 'H'
F2D3 32CEF1                 LD    (0F1CEH), A
F2D6 AF                     XOR   A
F2D7 32CFF1                 LD    (0F1CFH), A
F2DA 32F4F1                 LD    (0F1F4H), A
F2DD 32DEF1                 LD    (0F1DEH), A
F2E0 C34360                 JP    6043H

F2E3                SELRAM:
F2E3 F3                     DI
F2E4 3AC2E6                 LD    A, (PORT31)
F2E7 F602                   OR    2
F2E9 D331                   OUT   (31H), A
F2EB 7E                     LD    A, (HL)
F2EC C36CF1                 JP    0F16CH

F2EF                        END
```

付-2 CP/Mのファイル構造

PC-8801mkⅡでは第1章で述べたように汎用OS（基本ソフト）としてCP/Mが使用できます。ここではPC-8801mkⅡ用にカスタマイズされたCP/MとしてNECのCP/M Ver2.2のファイルとディレクトリの構造を述べます。CP/MとBASICとの間でファイルの転送などを行なうときに役に立つと思います。

1 ディスクマップ

1-1 サポートしているディスク

NECのCP/M Ver2.2（CBIOS Ver3.0）では5種類のディスクドライブを使用できます。

1．インテリジェント5インチディスクドライブ（両面／片面）
2．DMA 8インチディスクドライブ（PC-8881/82）
3．DMA 5インチディスクドライブ
4．ハードディスクドライブ（PC-98H31/32/33/34）
5．RAMディスクドライブ（128Kボード1 or 2 or 3）

このうちよく使われるいくつかのものについてのディスクマップを述べていきます。また、使われている用語の意味は次のとおりです。

ディレクトリエントリ

そのディスクに収納できるディレクトリの数です。ディレクトリが一杯になるとディスクにまだ空き領域があってもそれ以上ファイルを作ることができなくなります。

ブロック

CP/Mが実際にファイルを扱うときの単位です。BASICのクラスタにほぼ相当します。1ブロックより小さなファイルをつくることはできません。ディスクはブロックによって論理的に分割され、ディレクトリの最初を0として順にブロック番号が付けられています。

レコード

CP/Mがディスクとやりとりする論理的な最小単位で128バイトです。（BASICでは256バイト）

トラック

ここでも両面ディスクの場合、問題になるのがトラックです。物理的フォーマットでは1つのシリンダに表裏の2つのトラックがあると考えます。論理的フォーマットではもともと両面という考え方がなく、物理的に表裏にあるトラックを一続きのトラックだと考えます。

1-2　5－2D　フォーマットディスク（内蔵ドライブ）

物理的フォーマット

256バイト/セクタ

16セクタ/トラック

80トラック/ディスク

論理的フォーマット

64セクタ/トラック（シリンダ）

304Kバイト/ディスク

128ディレクトリエントリ

2システムトラック（シリンダ）

2Kバイト/ブロック

ディスクマップ

トラック	サーフェス	セクタ	内　容
0	0	1	IPL
0	0	2, 3	IPLの続き
0	0	4～16	BIOS
0	1	1～16	BIOS
1	0, 1	1～16	CCP&BDOS
2	0	1～16	ディレクトリ（ブロック#0, #1）
2	1	1～16	ユーザ領域
∫	∫	∫	∫　（ブロック#2～#151）
39	0, 1	1～16	ユーザ領域

1-3　5－1D　フォーマットディスク　(PC-8031/32)

物理的フォーマット

256バイト/セクタ

16セクタ/トラック

40トラック/ディスク

論理的フォーマット

32セクタ/トラック

152Kバイト/ディスク

64ディレクトリエントリ

2システムトラック

1Kバイト/ブロック

ディスクマップ

トラック	サーフェス	セクタ	内　　　容
0	0	1	IPL
0	0	2～16	CCP&BDOS
1	0	1～16	CCP&BDOS
2	0	1～8	ディレクトリ（ブロック＃0，＃1）
2	0	9～16	ユーザ領域
∫	∫	∫	∫　　（ブロック＃2～＃131）
39	0	1～16	ユーザ領域

BIOSはファイルとして存在します。

1-4　DMA 8インチ　片面単密　128バイト/セクタ　（PC-8881/82）

これはすべてのCP/Mに共通なフォーマットです。

物理的フォーマット

128バイト/セクタ

26セクタ/トラック

77トラック/ディスク

論理的フォーマット

26セクタ/トラック

243Kバイト/ディスク

64ディレクトリエントリ

2システムトラック

1Kバイト/ブロック

ディスクマップ

トラック	サーフェス	セクタ	内　　　容
0，1	0	1～26	未使用
2	0	1～8	ディレクトリ（ブロック＃0，＃1）
2	0	9～26	ユーザ領域
∫	∫	∫	∫　　（ブロック＃2～＃242）
76	0	1～26	ユーザ領域

システムを入れることはできません。

1-5　DMA 8インチ　両面倍密　256バイト/セクタ　(PC-8881/82)

物理的フォーマット

256バイト/セクタ

26セクタ/トラック

154トラック/ディスク

論理的フォーマット

104セクタ/トラック(シリンダ)

972Kバイト/ディスク

128ディレクトリエントリ

2システムトラック(シリンダ)

4Kバイト/ブロック

ディスクマップ

トラック	サーフェス	セクタ	内　　　容
0	0	1～26	未使用
0	1	1～26	CCP&BDOS
1	0	1	IPL
1	0	2,3	IPLの続き
1	0	4～26	BIOS
1	1	1～26	BIOS
2	0	1～16	ディレクトリ(ブロック#0)
2	0	17～26	ユーザ領域
∫	∫	∫	∫　(ブロック#1～#242)
76	0	1～26	ユーザ領域

1-5　RAMディスクドライブ　(PC-8801-02N)

論理的フォーマット

64セクタ/トラック

128Kバイト/ディスク(1枚あたり)

64ディレクトリエントリ

0システムトラック

2Kバイト/ブロック

システムを入れることはできません。

2　ディレクトリの構造

CP/Mでのファイル管理はすべてディレクトリによって行なわれ、BASICのようなFATは使われません。したがって、BASICではFATが持っているような情報もディレクトリが管理しています。一つのディレクトリは次のように32バイトで構成されます。

バイト	内　　容
0	ユーザ番号(00〜0F) ERASEされたファイルではE5Hが入る
1〜8	ファイル名(プライマリネーム)
9〜11	ファイル名（イクステンション）COM, ASM, PRNなど
12	ロジカルイクステント（続番）
13〜14	通常は0
15	総レコード数　そのディレクトリが管理しているレコード数(0〜80H)
16〜31	ディスクアロケーションマップ（使用しているブロック番号の並び）

ファイル名の「イクステンション」には通常7FH以下のコードが入っていますが、リードオンリーやシステムファイルに指定されると最上位ビットが立ちます。

CP/MにはFATがありませんので一つのディレクトリで管理できる領域には限りがあります。「ディスクアロケーションマップ（ブロック番号の並び）」は16バイトしかありません。また、「総レコード数」には80Hまでの値しか入りませんからここにも制限があります。これらを超えたらどうなるのかというと、次のようになります。

①「総レコード数」が128を越えたときには「ロジカルイクステント（続番)」を一つ増やし、「総レコード数」を０にします。

②ディスクアロケーションマップが一杯になった時は同じファイル名のディレクトリをもう一つ作って、そのディレクトリの「ロジカルイクステント」にはいままでのディレクトリのロジカルイクステントより一つ大きな値を書き込みます。新しいディレクトリの「総レコード数」は０になります。

ですから、大きなファイルを作ると、同じファイル名のディレクトリがたくさんできることになります。

付録-3　コントロールコード一覧表

(1) BASIC 入力時

16進	10進	対応するキー	N－BASIC	N_{88}－BASIC
0 1	1	CTRL－A		ヘルプキーと同じ
0 2	2	CTRL－B	1つ前のワードへ戻る	(N－BASICと同じ)
0 3	3	CTRL－C	実行の中断（STOP の時）	実行の中断
0 4	4	CTRL－D		カーソル位置から1ワードを削除
0 5	5	CTRL－E	カーソル位置から後を消す	(N－BASICと同じ)
0 6	6	CTRL－F		1つ先のワードへ進む
0 7	7	CTRL－G	スピーカを鳴らす	(N－BASICと同じ)
0 8	8	CTRL－H	カーソル位置の左側の文字を削除する	(N－BASICと同じ)
0 9	9	CTRL－I	水平タブ（8文字毎）	(N－BASICと同じ)
0 A	1 0	CTRL－J	行を2つに分ける	ラインフィード、インサートモードで2行に分割
0 B	1 1	CTRL－K	ホームポジション	(N－BASICと同じ)
0 C	1 2	CTRL－L	テキスト画面クリア	(N－BASICと同じ)
0 D	1 3	CTRL－M	キャリッジリターン	(N－BASICと同じ)
0 E	1 4	CTRL－N	1つ先のワードへ進む	
0 F	1 5	CTRL－O	ESC の後に押すことによりN_{88}－BASICと同じ働きを行なう	画面の表示を無効にする
1 2	1 8	CTRL－R	カーソル位置から右側を1文字分右へずらす。	インサートモードにする。
1 3	1 9	CTRL－S		実行を一時停止する
1 5	2 1	CTRL－U		1行キャンセル
1 8	2 4	CTRL－X		カーソルを行の最後に移す
1 B	2 7	ESC	実行を一時停止する	
1 C	2 8	→	カーソルを右へ移動	(N－BASICと同じ)
1 D	2 9	←	〃　左　〃	(N－BASICと同じ)
1 E	3 0	↑	〃　上　〃	(N－BASICと同じ)
1 F	3 1	↓	〃　下　〃	(N－BASICと同じ)

(2) 通信用（ASCII）

16進	10進	シンボル	シンボルの意味
0 0	0		null
0 1	1	SH	Start of Heading（ヘッディング開始）
0 2	2	SX	Start of Text（テキスト開始）
0 3	3	EX	End of Text（テキスト終了）
0 4	4	ET	End of Transmission（伝送終了）
0 5	5	EQ	Enquiry（問合わせ）
0 6	6	AK	Acknowledge（肯定応答）
0 7	7	BL	Bell（ベル、ブザー）
0 8	8	BS	Back Space（後退）
0 9	9	HT	Horizontal Tabulation（水平タブ）
0 A	1 0	LF	Line Feed（改行）
0 B	1 1	HM	Home(VT) Vertical Tabulation（垂直タブ）
0 C	1 2	CL	Clear(FF) Form Feed（改頁）
0 D	1 3	CR	Carriage Return（復帰）
0 E	1 4	SO	Sift-out（シフトアウト）
0 F	1 5	SI	Sift-in（シフトイン）
1 0	1 6	DE	Data Link Escape（伝送制御拡張）
1 1	1 7	D1	Device Control1（装置制御1）
1 2	1 8	D2	Device Control2（装置制御2）
1 3	1 9	D3	Device Control3（装置制御3）
1 4	2 0	D4	Device Control4（装置制御4）
1 5	2 1	NK	Negative Acknowledge（否定応答）
1 6	2 2	SN	Synchronous idle（同期信号）
1 7	2 3	EB	End of Transmission Block（伝送ブロック終了）
1 8	2 4	CN	Cancel（取消し）
1 9	2 5	EM	End of Medium（媒体終端）
1 A	2 6	SB	Substitute（文字置換）
1 B	2 7	EC	Escape（拡張）
1 C	2 8	→	(FS)File Separator（ファイル分離）
1 D	2 9	←	(GS) Group Separator（グループ分離）
1 E	3 0	↑	(RS)Record Separator（レコード分離）
1 F	3 1	↓	(US)Unit Separator（ユニット分離）

付録-4　ＥＢＣＤＩＣコード

ＥＢＣＤＩＣ（カナ入り）コード表

上位４ビット→

下位４ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	－			ソ			{	}		0
1	SOH	DC1	SOS		｡	ェ	／		a ア	j タ			A	J		1
2	STX	DC2	FS	SYN	｢	ォ			b イ	k チ	s ヘ		B	K	S	2
3	ETX	TM			｣	ャ			c ウ	l ツ	t ホ		C	L	T	3
4	PF	RES	BYP	PN		ュ			d エ	m テ	u マ		D	M	U	4
5	HT	NL	LF	RS	・	ョ			e オ	n ト	v ミ		E	N	V	5
6	LC	BS	ETB	UC	ヲ	ッ			f カ	o ナ	w ム		F	O	W	6
7	DL	IL	ESC	EOT	ァ				g キ	p ニ	x メ		G	P	X	7
8	GE	CAN			ィ	－			h ク	q ヌ	y モ		H	Q	Y	8
9	RLF	EM			ゥ				i ケ	r ネ	z ヤ		I	R	Z	9
A	SMM	CC	SM		[	!	]	:	コ	ノ	ユ	レ				
B	VT	CU1	CU2	CU3	.	$	,	#				ロ				
C	FF	IFS		DC4	<	*	%	@	サ		ヨ	ワ				
D	CR	IGS	ENR	NAK	(	)	－	'	シ	ハ	ラ	ン				
E	SO	IRS	ACK		+	;	>	=	ス	ヒ	リ	゛				
F	SI	IUS	BEL	SUB	|	¬	?	"	セ	フ	ル	゜				

付録-5　演算順位表

演算順位表と誘導関数表

演算順位表

順　位	演　算　の　種　類
1	(　　)で囲まれたもの
2	関数
3	指数(べき乗)……A^2(A^2)
4	負号(－)
5	*／
6	¥
7	MOD
8	＋，－
9	条件判定(<，>，＝など)
10	NOT
11	AND
12	OR
13	XOR
14	IMP
15	EQV

注）演算の順位が同じ場合，左から右へ順に行なわれます。

誘導関数表

目的とする関数	組み込み関数からの誘導式
セカント	SEC(X)=1/COS(X)
コセカント	CSC(X)=1/SIN(X)
コタンジェント	COT(X)=1/TAN(X)
アークサイン	ARCSIN(X)=ATN(X/SQR(−X*X+1))
アークコサイン	ARCCOS(X)=−ATN(X/SQR(−X*X+1))+1.5708
アークセカント	ARCSEC(X)=ATN(SQR(X*X−1))+(SGN(X)−1*1.5708
アークコセカント	ARCCSC(X)=ATN(1/SQR(X*X−1))+(SGN(N(X)−1)*1.5708
アークコタンジェント	ARCCOT(X)=−ATN(N)+1.5708
ハイパーボリック・サイン	SINH(X)=(EXP(X)−EXP(−X))/2
ハイパーボリック・コサイン	COSH(X)=(EXP(X)+EXP(−X))/2
ハイパーボリック・タンジェント	TANH(X)=−EXP(−X)/(EXP(X)+EXP(−X))*2+1
ハイパーボリック・セカント	SECH(X)=2/(EXP(X)+EXP(−X))
ハイパーボリック・コセカント	CSCH(X)=2/(EXP(X)−EXP(−X))
ハイパーボリック・コタンジェント	COTH(X)=EXP(−X)/(EXP(X)−EXP(−X))*2+1
ハイパーボリック・アークサイン	ARCSINH(X)=LOG(X+SQR(X*X+1))
ハイパーボリック・アークコサイン	ARCCOSH(X)=LOG(X+SQR(X*X−1))
ハイパーボリック・アークタンジェント	ARCTANH(X)=LOG((1+X)/(1−X))/2
ハイパーボリック・アークセカント	ARCSECH(X)=LOG((SQR(−X*X+1)+1)+1)/X)
ハイパーボリック・アークコセカント	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
ハイパーボリック・アークコタンジェント	ARCCOTH(X)=LOG((X+1)/(X−1))/2

付録-6　USING文フォーマット一覧表

フォーマット	機　　能	例
!	文字列の最初の文字だけ出力	PRINT USING "F.1(!)";"abcde" F.1(a)
&　・・　& n文字	始めのn文字を左づめで出力	PRINT USING "F.2:(&　&)";"abcde" F.2:(abcd)
@	1つの@に対して、1つの文字列を出力	PRINT USING "F.3:(@)";"abcde" F.3:(abcde)
＃＃＃＃	数値を右づめで表示	PRINT USING "F.4:(####)";123.456 F.4:(123)
＃＃＃＃．＃	小数点の位置を指定	PRINT USING "F.5:(####.#)";123.456 F.5:(123.5)
＋＃＃＃．＃	数値の前に符号(＋，－)をつける	PRINT USING "F.6:(+###.#)";123.456 F.6:(+123.5)
＃＃＃．＃＋	数値の後に符号(＋，－)をつける	PRINT USING "F.7:(###.#+)";123.456,-123.456 F.7:(123.5+)F.7:(123.5-) PRINT USING "F.8:(###.#-)";123.456,-123.456 F.8:(123.5)F.8:(123.5-)
＊＊＃＃＃＃．＃	空白部分を“＊”で埋める	PRINT USING "F.9:(**####.#)";123.456 F.9:(***123.5)
￥￥＃＃＃＃．＃	数値の直前に“￥”をつける	PRINT USING "F.10:(￥￥###.#)";123.456 F.10:(￥123.5)
＊＊￥＃＃．＃	＊＊と￥￥の両方の機能となる	PRINT USING "F.11:(**￥##.#)";123.456 F.11:(*￥123.5)
＃＃＃＃＃，	3桁毎に“，”で区切って出力	PRINT USING "F.12:(#####,)";1234.56 F.12:(　1,235)
＃＃＃^^^^	数値を指数形式で出力	PRINT USING "F.13:(###^^^^)";1234.56 F.13:(12E+02)
＃＃＃＃_，＃＃＃＃	“_”に続く1文字を単に文字として出力	PRINT USING "F.14:(####_,####)";123,123 F.14:(123, 123)

付-7 エラーメッセージ一覧表

DECI	HEX	-ERROR MESSAGE-
1	01	NEXT without FOR
2	02	Syntax error
3	03	RETURN without GOSUB
4	04	Out of DATA
5	05	Illegal function call
6	06	Overflow
7	07	Out of memory
8	08	Undefined line number
9	09	Subscript out of range
10	0A	Duplicate Definition
11	0B	Division by zero
12	0C	Illegal direct
13	0D	Type mismatch
14	0E	Out of string space
15	0F	String too long
16	10	String formula too complex
17	11	Can't continue
18	12	Undefined user function
19	13	No RESUME
20	14	RESUME without error
21	15	Unprintable error
22	16	Missing operand
23	17	Line buffer overflow
24	18	?
25	19	?
26	1A	FOR Without NEXT
27	1B	Tape read ERROR
28	1C	?
29	1D	WHILE without WEND
30	1E	WEND without WHILE
31	1F	duplicate label
32	20	undefined label
33	21	Feature not available
50	32	FIELD overflow
51	33	Internal error
52	34	Bad file number
53	35	File not found
54	36	File already open
55	37	Input past end
56	38	Bad file name
57	39	Direct statement in file
58	3A	Sequential after PUT
59	3B	Sequential I/O only
60	3C	File not OPEN
61	3D	File write protected
62	3E	Disk offline
63	3F	Disk not mounted
64	40	Disk I/O error
65	41	File already exists
66	42	?

67	43	Disk already mounted
68	44	Disk full
69	45	Bad allocation table
70	46	Bad drive number
71	47	Bad track/sector
72	48	Deleted record
73	49	Rename across disks

N_{88}-ROM-BASIC, N_{88}-DISK-BASICエラーメッセージ対応表

AO	54	File already open
BN	52	Bad file number
BO	23	Line buffer overflow
BS	9	Subscript out of range
CF	60	File not OPEN
CN	17	Can't continue
DD	10	Duplicate Definition
DS	57	Direct statement in file
DU	31	duplicate label
EF	55	Input past end
FC	5	Illegal function call
FF	53	File not found
FN	26	FOR Without NEXT
FO	50	FIELD overflow
ID	12	Illegal direct
IE	51	Internal error
LS	15	String too long
MO	22	Missing operand
NA	33	Feature not available
NF	1	NEXT without FOR
NM	56	Bad file name
NR	19	No RESUME
OD	4	Out of DATA
OM	7	Out of memory
OS	14	Out of string space
OV	6	Overflow
RG	3	RETURN without GOSUB
RW	20	RESUME without error
SN	2	Syntax error
SP	58	Sequential after PUT
SQ	59	Sequential I/O only
ST	16	String formula too complex
TM	13	Type mismatch
TP	27	Tape read ERROR
UE	21	Unprintable error
UF	18	Undefined user function
UL	8	Undefined line number
UN	32	undefined label
WE	30	WEND without WHILE
WH	29	WHILE without WEND
/0	11	Division by zero

付-8 漢字キャラクタ対応表

使用プリンタ：NM9400

```
〔 キゴウ 〕-----------------------------------------------------------
   =!!  、=!｢  。=!#  ,=!$  .=!%  ・=!&  :=!'  ;=!〔  ?=!〕  !=!*
゛=!+  ゜=!,  ´=!-  `=!.  ¨=!/  ^=!0  ￣=!1  _=!2  ヽ=!3  ヾ=!4
ゝ=!5  ゞ=!6  〃=!7  仝=!8  々=!9  〆=!:  〇=!;  ー=!<  ―=!=  -=!>
/=!?  \=!@  ~=!A  ‖=!B  |=!C  …=!D  ‥=!E  ‘=!F  ’=!G  “=!H
”=!I  (=!J  )=!K  〔=!L  〕=!M  [=!N  ]=!O  {=!P  }=!Q  〈=!R
〉=!S  《=!T  》=!U  「=!V  」=!W  『=!X  』=!Y  【=!Z  】=!〔  +=!¥
-=!〕  ±=!^  ×=!_  ÷=!=  ==!a  ≠=!b  <=!c  >=!d  ≦=!e  ≧=!f
∞=!g  ∴=!h  ♂=!i  ♀=!j  °=!k  ′=!l  ″=!m  ℃=!n  ¥=!o  $=!p
¢=!q  £=!r  %=!s  #=!t  &=!u  *=!v  @=!w  §=!x  ☆=!y  ★=!z
○=!{  ●=!¦  ◎=!}  ◇=!~  ◆=｢!  □=｢｢  ■=｢#  △=｢$  ▲=｢%  ▽=｢&
▼=｢'  ※=｢〔  〒=｢〕  →=｢*  ←=｢+  ↑=｢,  ↓=｢-  〓=｢.   =｢/   =｢0
   =｢1   =｢2   =｢3   =｢4   =｢5   =｢6   =｢7   =｢8   =｢9  ∈=｢:
∋=｢;  ⊆=｢<  ⊇=｢=  ⊂=｢>  ⊃=｢?  ∪=｢@  ∩=｢A   =｢B   =｢C   =｢D
   =｢E   =｢F   =｢G   =｢H   =｢I  ∧=｢J  ∨=｢K  ¬=｢L  ⇒=｢M  ⇔=｢N
∀=｢O  ∃=｢P   =｢Q   =｢R   =｢S   =｢T   =｢U   =｢V   =｢W   =｢X
   =｢Y   =｢Z   =｢〔  ∠=｢¥  ⊥=｢〕  ⌒=｢^  ∂=｢_  ∇=｢=  ≡=｢a  ≒=｢b
≪=｢c  ≫=｢d  √=｢e  ∽=｢f  ∝=｢g  ∵=｢h  ∫=｢i  ∬=｢j   =｢k   =｢l
   =｢m   =｢n   =｢o   =｢p   =｢q  Å=｢r  ‰=｢s  ♯=｢t  ♭=｢u  ♪=｢v
†=｢w  ‡=｢x  ¶=｢y   =｢z   =｢{   =｢¦   =｢}  ◯=｢~   =#!   =#｢
   =##   =#$   =#%   =#&   =#'   =#〔   =#〕   =#*   =#+   =#,
   =#-   =#.   =#/  0=#0  1=#1  2=#2  3=#3  4=#4  5=#5  6=#6
7=#7  8=#8  9=#9   =#:   =#;   =#<   =#=   =#>   =#?   =#@
A=#A  B=#B  C=#C  D=#D  E=#E  F=#F  G=#G  H=#H  I=#I  J=#J
K=#K  L=#L  M=#M  N=#N  O=#O  P=#P  Q=#Q  R=#R  S=#S  T=#T
U=#U  V=#V  W=#W  X=#X  Y=#Y  Z=#Z   =#〔   =#¥   =#〕   =#^
   =#_   =#=  a=#a  b=#b  c=#c  d=#d  e=#e  f=#f  g=#g  h=#h
i=#i  j=#j  k=#k  l=#l  m=#m  n=#n  o=#o  p=#p  q=#q  r=#r
s=#s  t=#t  u=#u  v=#v  w=#w  x=#x  y=#y  z=#z   =#{   =#¦
   =#}   =#~  ぁ=$!  あ=$｢  ぃ=$#  い=$$  ぅ=$%  う=$&  ぇ=$'  え=$〔
ぉ=$〕  お=$*  か=$+  が=$,  き=$-  ぎ=$.  く=$/  ぐ=$0  け=$1  げ=$2
こ=$3  ご=$4  さ=$5  ざ=$6  し=$7  じ=$8  す=$9  ず=$:  せ=$;  ぜ=$<
そ=$=  ぞ=$>  た=$?  だ=$@  ち=$A  ぢ=$B  っ=$C  つ=$D  づ=$E  て=$F
で=$G  と=$H  ど=$I  な=$J  に=$K  ぬ=$L  ね=$M  の=$N  は=$O  ば=$P
ぱ=$Q  ひ=$R  び=$S  ぴ=$T  ふ=$U  ぶ=$V  ぷ=$W  へ=$X  べ=$Y  ぺ=$Z
ほ=$〔  ぼ=$¥  ぽ=$〕  ま=$^  み=$_  む=$=  め=$a  も=$b  ゃ=$c  や=$d
ゅ=$e  ゆ=$f  ょ=$g  よ=$h  ら=$i  り=$j  る=$k  れ=$l  ろ=$m  ゎ=$n
わ=$o  ゐ=$p  ゑ=$q  を=$r  ん=$s   =$t   =$u   =$v   =$w   =$x
   =$y   =$z   =${   =$¦   =$}   =$~  ァ=%!  ア=%｢  ィ=%#  イ=%$
ゥ=%%  ウ=%&  ェ=%'  エ=%〔  ォ=%〕  オ=%*  カ=%+  ガ=%,  キ=%-  ギ=%.
ク=%/  グ=%0  ケ=%1  ゲ=%2  コ=%3  ゴ=%4  サ=%5  ザ=%6  シ=%7  ジ=%8
ス=%9  ズ=%:  セ=%;  ゼ=%<  ソ=%=  ゾ=%>  タ=%?  ダ=%@  チ=%A  ヂ=%B
ッ=%C  ツ=%D  ヅ=%E  テ=%F  デ=%G  ト=%H  ド=%I  ナ=%J  ニ=%K  ヌ=%L
ネ=%M  ノ=%N  ハ=%O  バ=%P  パ=%Q  ヒ=%R  ビ=%S  ピ=%T  フ=%U  ブ=%V
プ=%W  ヘ=%X  ベ=%Y  ペ=%Z  ホ=%〔  ボ=%¥  ポ=%〕  マ=%^  ミ=%_  ム=%=
メ=%a  モ=%b  ャ=%c  ヤ=%d  ュ=%e  ユ=%f  ョ=%g  ヨ=%h  ラ=%i  リ=%j
ル=%k  レ=%l  ロ=%m  ヮ=%n  ワ=%o  ヰ=%p  ヱ=%q  ヲ=%r  ン=%s  ヴ=%t
ヵ=%u  ヶ=%v   =%w   =%x   =%y   =%z   =%{   =%¦   =%}   =%~
Α=&!  Β=&｢  Γ=&#  Δ=&$  Ε=&%  Ζ=&&  Η=&'  Θ=&〔  Ι=&〕  Κ=&*
Λ=&+  Μ=&,  Ν=&-  Ξ=&.  Ο=&/  Π=&0  Ρ=&1  Σ=&2  Τ=&3  Υ=&4
Φ=&5  Χ=&6  Ψ=&7  Ω=&8   =&9   =&:   =&;   =&<   =&=   =&>
   =&?   =&@  α=&A  β=&B  γ=&C  δ=&D  ε=&E  ζ=&F  η=&G  θ=&H
ι=&I  κ=&J  λ=&K  μ=&L  ν=&M  ξ=&N  ο=&O  π=&P  ρ=&Q  σ=&R
```

τ=&S υ=&T φ=&U χ=&V ψ=&W ω=&X =&Y =&Z =&〔 =&¥
=&〕 =&^ =&_ =&= =&a =&b =&c =&d =&e =&f
=&g =&h =&i =&j =&k =&l =&m =&n =&o =&p
=&q =&r =&s =&t =&u =&v =&w =&x =&y =&z
=&{ =&¦ =&} =&~ А='! Б='「 В='# Г='$ Д='% Е='&
Ё='' Ж='(З=') И='* Й='+ К=', Л='- М='. Н='/ О='0
П='1 Р='2 С='3 Т='4 У='5 Ф='6 Х='7 Ц='8 Ч='9 Ш=':
Щ='; Ъ='< Ы='= Ь='> Э='? Ю='@ Я='A ='B ='C ='D
='E ='F ='G ='H ='I ='J ='K ='L ='M ='N
='O ='P а='Q б='R в='S г='T д='U е='V ё='W ж='X
з='Y и='Z й='〔 к='¥ л='〕 м='^ н='_ о='= п='a р='b
с='c т='d у='e ф='f х='g ц='h ч='i ш='j щ='k ъ='l
ы='m ь='n э='o ю='p

〔ア〕 --

亜=0! 唖=0「 娃=0# 阿=0$ 哀=0% 愛=0& 挨=0' 姶=0(逢=0) 葵=0*
茜=0+ 穐=0, 悪=0- 握=0. 渥=0/ 旭=00 葦=01 芦=02 鯵=03 梓=04
圧=05 斡=06 扱=07 宛=08 姐=09 虻=0: 飴=0; 絢=0< 綾=0= 鮎=0>
或=0? 粟=0@ 袷=0A 安=0B 庵=0C 按=0D 暗=0E 案=0F 闇=0G 鞍=0H
杏=0I

〔イ〕 --

以=0J 伊=0K 位=0L 依=0M 偉=0N 囲=0O 夷=0P 委=0Q 威=0R 尉=0S
惟=0T 意=0U 慰=0V 易=0W 椅=0X 為=0Y 畏=0Z 異=0〔 移=0¥ 維=0〕
緯=0^ 胃=0_ 萎=0= 衣=0a 謂=0b 違=0c 遺=0d 医=0e 井=0f 亥=0g
域=0h 育=0i 郁=0j 磯=0k 一=0l 壱=0m 溢=0n 逸=0o 稲=0p 茨=0q
芋=0r 鰯=0s 允=0t 印=0u 咽=0v 員=0w 因=0x 姻=0y 引=0z 飲=0{
淫=0¦ 胤=0} 蔭=0~ 院=1! 陰=1「 隠=1# 韻=1$ 吋=1%

〔ウ〕 --

右=1& 宇=1' 烏=1(羽=1) 迂=1* 雨=1+ 卯=1, 鵜=1- 窺=1. 丑=1/
碓=10 臼=11 渦=12 嘘=13 唄=14 欝=15 蔚=16 鰻=17 姥=18 厩=19
浦=1: 瓜=1; 閏=1< 噂=1= 云=1> 運=1? 雲=1@

〔エ〕 --

荏=1A 餌=1B 叡=1C 営=1D 嬰=1E 影=1F 映=1G 曳=1H 栄=1I 永=1J
泳=1K 洩=1L 瑛=1M 盈=1N 穎=1O 頴=1P 英=1Q 衛=1R 詠=1S 鋭=1T
液=1U 疫=1V 益=1W 駅=1X 悦=1Y 謁=1Z 越=1〔 閲=1¥ 榎=1〕 厭=1^
円=1_ 園=1= 堰=1a 奄=1b 宴=1c 延=1d 怨=1e 掩=1f 援=1g 沿=1h
演=1i 炎=1j 焔=1k 煙=1l 燕=1m 猿=1n 縁=1o 艶=1p 苑=1q 薗=1r
遠=1s 鉛=1t 鴛=1u 塩=1v

〔オ〕 --

於=1w 汚=1x 甥=1y 凹=1z 央=1{ 奥=1¦ 往=1} 応=1~ 押=2! 旺=2「
横=2# 欧=2$ 殴=2% 王=2& 翁=2' 襖=2(鴬=2) 鴎=2* 黄=2+ 岡=2,
沖=2- 荻=2. 億=2/ 屋=20 憶=21 臆=22 桶=23 牡=24 乙=25 俺=26
卸=27 恩=28 温=29 穏=2: 音=2;

〔カ〕 --

下=2< 化=2= 仮=2> 何=2? 伽=2@ 価=2A 佳=2B 加=2C 可=2D 嘉=2E
夏=2F 嫁=2G 家=2H 寡=2I 科=2J 暇=2K 果=2L 架=2M 歌=2N 河=2O
火=2P 珂=2Q 禍=2R 禾=2S 稼=2T 箇=2U 花=2V 苛=2W 茄=2X 荷=2Y
華=2Z 菓=2〔 蝦=2¥ 課=2〕 嘩=2^ 貨=2_ 迦=2= 過=2a 霞=2b 蚊=2c
俄=2d 峨=2e 我=2f 牙=2g 画=2h 臥=2i 芽=2j 蛾=2k 賀=2l 雅=2m
餓=2n 駕=2o 介=2p 会=2q 解=2r 回=2s 塊=2t 壊=2u 廻=2v 快=2w
怪=2x 悔=2y 恢=2z 懐=2{ 戒=2¦ 拐=2} 改=2~ 魁=3! 晦=3「 械=3#
海=3$ 灰=3% 界=3& 皆=3' 絵=3(芥=3) 蟹=3* 開=3+ 階=3, 貝=3-
凱=3. 劾=3/ 外=30 咳=31 害=32 崖=33 慨=34 概=35 涯=36 碍=37
蓋=38 街=39 該=3: 鎧=3; 骸=3< 浬=3= 馨=3> 蛙=3? 垣=3@ 柿=3A
蛎=3B 鈎=3C 劃=3D 嚇=3E 各=3F 廓=3G 拡=3H 撹=3I 格=3J 核=3K
殻=3L 獲=3M 確=3N 穫=3O 覚=3P 角=3Q 赫=3R 較=3S 郭=3T 閣=3U
隔=3V 革=3W 学=3X 岳=3Y 楽=3Z 額=3〔 顎=3¥ 掛=3〕 笠=3^ 樫=3_

橿=3＝ 梶=3a 鰍=3b 潟=3c 割=3d 喝=3e 恰=3f 括=3g 活=3h 渇=3i
滑=3j 葛=3k 褐=3l 轄=3m 且=3n 鰹=3o 叶=3p 椛=3q 樺=3r 鞄=3s
株=3t 兜=3u 竃=3v 蒲=3w 釜=3x 鎌=3y 噛=3z 鴨=3{ 栢=3¦ 茅=3}
萱=3~ 粥=4! 刈=4「 苅=4# 瓦=4$ 乾=4% 侃=4& 冠=4' 寒=4(刊=4)
勘=4* 勧=4+ 巻=4, 喚=4- 堪=4. 姦=4/ 完=40 官=41 寛=42 干=43
幹=44 患=45 感=46 慣=47 憾=48 換=49 敢=4: 柑=4; 桓=4< 棺=4=
款=4> 歓=4? 汗=4@ 漢=4A 澗=4B 潅=4C 環=4D 甘=4E 監=4F 看=4G
竿=4H 管=4I 簡=4J 緩=4K 缶=4L 翰=4M 肝=4N 艦=4O 莞=4P 観=4Q
諌=4R 貫=4S 還=4T 鑑=4U 間=4V 閑=4W 関=4X 陥=4Y 韓=4Z 館=4〔
舘=4¥ 丸=4〕 含=4^ 岸=4_ 巌=4＝ 玩=4a 癌=4b 眼=4c 岩=4d 翫=4e
贋=4f 雁=4g 頑=4h 顔=4i 願=4j

〔キ〕 --

企=4k 伎=4l 危=4m 喜=4n 器=4o 基=4p 奇=4q 嬉=4r 寄=4s 岐=4t
希=4u 幾=4v 忌=4w 揮=4x 机=4y 旗=4z 既=4{ 期=4¦ 棋=4} 棄=4~
機=5! 帰=5「 毅=5# 気=5$ 汽=5% 畿=5& 祈=5' 季=5(稀=5) 紀=5*
徽=5+ 規=5, 記=5- 貴=5. 起=5/ 軌=50 輝=51 飢=52 騎=53 鬼=54
亀=55 偽=56 儀=57 妓=58 宜=59 戯=5: 技=5; 擬=5< 欺=5= 犠=5>
疑=5? 祇=5@ 義=5A 蟻=5B 誼=5C 議=5D 掬=5E 菊=5F 鞠=5G 吉=5H
吃=5I 喫=5J 桔=5K 橘=5L 詰=5M 砧=5N 杵=5O 黍=5P 却=5Q 客=5R
脚=5S 虐=5T 逆=5U 丘=5V 久=5W 仇=5X 休=5Y 及=5Z 吸=5〔 宮=5¥
弓=5〕 急=5^ 救=5_ 朽=5＝ 求=5a 汲=5b 泣=5c 灸=5d 球=5e 究=5f
窮=5g 笈=5h 級=5i 糾=5j 給=5k 旧=5l 牛=5m 去=5n 居=5o 巨=5p
拒=5q 拠=5r 挙=5s 渠=5t 虚=5u 許=5v 距=5w 鋸=5x 漁=5y 禦=5z
魚=5{ 亨=5¦ 享=5} 京=5~ 供=6! 侠=6「 僑=6# 兇=6$ 競=6% 共=6&
凶=6' 協=6(匡=6) 卿=6* 叫=6+ 喬=6, 境=6- 峡=6. 強=6/ 彊=60
怯=61 恐=62 恭=63 挟=64 教=65 橋=66 況=67 狂=68 狭=69 矯=6:
胸=6; 脅=6< 興=6= 蕎=6> 郷=6? 鏡=6@ 響=6A 饗=6B 驚=6C 仰=6D
凝=6E 尭=6F 暁=6G 業=6H 局=6I 曲=6J 極=6K 玉=6L 桐=6M 粁=6N
僅=6O 勤=6P 均=6Q 巾=6R 錦=6S 斤=6T 欣=6U 欽=6V 琴=6W 禁=6X
禽=6Y 筋=6Z 緊=6〔 芹=6¥ 菌=6〕 衿=6^ 襟=6_ 謹=6＝ 近=6a 金=6b
吟=6c 銀=6d

〔ク〕 --

九=6e 倶=6f 句=6g 区=6h 狗=6i 玖=6j 矩=6k 苦=6l 躯=6m 駆=6n
駈=6o 駒=6p 具=6q 愚=6r 虞=6s 喰=6t 空=6u 偶=6v 寓=6w 遇=6x
隅=6y 串=6z 櫛=6{ 釧=6¦ 屑=6} 屈=6~ 掘=7! 窟=7「 沓=7# 靴=7$
轡=7% 窪=7& 熊=7' 隈=7(粂=7) 栗=7* 繰=7+ 桑=7, 鍬=7- 勲=7.
君=7/ 薫=70 訓=71 群=72 軍=73 郡=74

〔ケ〕 --

卦=75 袈=76 祁=77 係=78 傾=79 刑=7: 兄=7; 啓=7< 圭=7= 珪=7>
型=7? 契=7@ 形=7A 径=7B 恵=7C 慶=7D 慧=7E 憩=7F 掲=7G 携=7H
敬=7I 景=7J 桂=7K 渓=7L 畦=7M 稽=7N 系=7O 経=7P 継=7Q 繋=7R
罫=7S 茎=7T 荊=7U 蛍=7V 計=7W 詣=7X 警=7Y 軽=7Z 頚=7〔 鶏=7¥
芸=7〕 迎=7^ 鯨=7_ 劇=7＝ 戟=7a 撃=7b 激=7c 隙=7d 桁=7e 傑=7f
欠=7g 決=7h 潔=7i 穴=7j 結=7k 血=7l 訣=7m 月=7n 件=7o 倹=7p
倦=7q 健=7r 兼=7s 券=7t 剣=7u 喧=7v 圏=7w 堅=7x 嫌=7y 建=7z
憲=7{ 懸=7¦ 拳=7} 捲=7~ 検=8! 権=8「 牽=8# 犬=8$ 献=8% 研=8&
硯=8' 絹=8(県=8) 肩=8* 見=8+ 謙=8, 賢=8- 軒=8. 遣=8/ 鍵=80
険=81 顕=82 験=83 鹸=84 元=85 原=86 厳=87 幻=88 弦=89 減=8:
源=8; 玄=8< 現=8= 絃=8> 舷=8? 言=8@ 諺=8A 限=8B

〔コ〕 --

乎=8C 個=8D 古=8E 呼=8F 固=8G 姑=8H 孤=8I 己=8J 庫=8K 弧=8L
戸=8M 故=8N 枯=8O 湖=8P 狐=8Q 糊=8R 袴=8S 股=8T 胡=8U 菰=8V
虎=8W 誇=8X 跨=8Y 鈷=8Z 雇=8〔 顧=8¥ 鼓=8〕 五=8^ 互=8_ 伍=8＝
午=8a 呉=8b 吾=8c 娯=8d 後=8e 御=8f 悟=8g 梧=8h 檎=8i 瑚=8j
碁=8k 語=8l 誤=8m 護=8n 醐=8o 乞=8p 鯉=8q 交=8r 佼=8s 侯=8t
候=8u 倖=8v 光=8w 公=8x 功=8y 効=8z 勾=8{ 厚=8¦ 口=8} 向=8~
后=9! 喉=9「 坑=9# 垢=9$ 好=9% 孔=9& 孝=9' 宏=9(工=9) 巧=9*

巷=9+ 幸=9, 広=9- 庚=9. 康=9/ 弘=90 恒=91 慌=92 抗=93 拘=94
控=95 攻=96 昂=97 晃=98 更=99 杭=9: 校=9; 梗=9< 構=9= 江=9>
洪=9? 浩=9@ 港=9A 溝=9B 甲=9C 皇=9D 硬=9E 稿=9F 糠=9G 紅=9H
紘=9I 絞=9J 綱=9K 耕=9L 考=9M 肯=9N 肱=9O 腔=9P 膏=9Q 航=9R
荒=9S 行=9T 衡=9U 講=9V 貢=9W 購=9X 郊=9Y 酵=9Z 鉱=9〔 礦=9¥
鋼=9〕 閤=9^ 降=9_ 項=9= 香=9a 高=9b 鴻=9c 剛=9d 劫=9e 号=9f
合=9g 壕=9h 拷=9i 濠=9j 豪=9k 轟=9l 麹=9m 克=9n 刻=9o 告=9p
国=9q 穀=9r 酷=9s 鵠=9t 黒=9u 獄=9v 漉=9w 腰=9x 甑=9y 忽=9z
惚=9{ 骨=9¦ 狛=9} 込=9~ 此=:! 頃=:「 今=:# 困=:$ 坤=:% 墾=:&
婚=:' 恨=:(懇=:) 昏=:* 昆=:+ 根=:, 梱=:- 混=:. 痕=:/ 紺=:0
艮=:1 魂=:2

〔サ〕 ---

些=:3 佐=:4 叉=:5 唆=:6 嵯=:7 左=:8 差=:9 査=:: 沙=:; 瑳=:<
砂=:= 詐=:> 鎖=:? 裟=:@ 坐=:A 座=:B 挫=:C 債=:D 催=:E 再=:F
最=:G 哉=:H 塞=:I 妻=:J 宰=:K 彩=:L 才=:M 採=:N 栽=:O 歳=:P
済=:Q 災=:R 采=:S 犀=:T 砕=:U 砦=:V 祭=:W 斎=:X 細=:Y 菜=:Z
裁=:〔 載=:¥ 際=:〕 剤=:^ 在=:_ 材=:= 罪=:a 財=:b 冴=:c 坂=:d
阪=:e 堺=:f 榊=:g 肴=:h 咲=:i 崎=:j 埼=:k 碕=:l 鷺=:m 作=:n
削=:o 咋=:p 搾=:q 昨=:r 朔=:s 柵=:t 窄=:u 策=:v 索=:w 錯=:x
桜=:y 鮭=:z 笹=:{ 匙=:¦ 冊=:} 刷=:~ 察=;! 拶=;「 撮=;# 擦=;$
札=;% 殺=;& 薩=;' 雑=;(皐=;) 鯖=;* 捌=;+ 錆=;, 鮫=;- 皿=;.
晒=;/ 三=;0 傘=;1 参=;2 山=;3 惨=;4 撒=;5 散=;6 桟=;7 燦=;8
珊=;9 産=;: 算=;; 纂=;< 蚕=;= 讃=;> 賛=;? 酸=;@ 餐=;A 斬=;B
暫=;C 残=;D

〔シ〕 ---

仕=;E 仔=;F 伺=;G 使=;H 刺=;I 司=;J 史=;K 嗣=;L 四=;M 士=;N
始=;O 姉=;P 姿=;Q 子=;R 屍=;S 市=;T 師=;U 志=;V 思=;W 指=;X
支=;Y 孜=;Z 斯=;〔 施=;¥ 旨=;〕 枝=;^ 止=;_ 死=;= 氏=;a 獅=;b
祉=;c 私=;d 糸=;e 紙=;f 紫=;g 肢=;h 脂=;i 至=;j 視=;k 詞=;l
詩=;m 試=;n 誌=;o 諮=;p 資=;q 賜=;r 雌=;s 飼=;t 歯=;u 事=;v
似=;w 侍=;x 児=;y 字=;z 寺=;{ 慈=;¦ 持=;} 時=;~ 次=<! 滋=<「
治=<# 爾=<$ 璽=<% 痔=<& 磁=<' 示=<(而=<) 耳=<* 自=<+ 蒔=<,
辞=<- 汐=<. 鹿=</ 式=<0 識=<1 鴫=<2 竺=<3 軸=<4 宍=<5 雫=<6
七=<7 叱=<8 執=<9 失=<: 嫉=<; 室=<< 悉=<= 湿=<> 漆=<? 疾=<@
質=<A 実=<B 蔀=<C 篠=<D 偲=<E 柴=<F 芝=<G 屡=<H 蕊=<I 縞=<J
舎=<K 写=<L 射=<M 捨=<N 赦=<O 斜=<P 煮=<Q 社=<R 紗=<S 者=<T
謝=<U 車=<V 遮=<W 蛇=<X 邪=<Y 借=<Z 勺=<〔 尺=<¥ 杓=<〕 灼=<^
爵=<_ 酌=<= 釈=<a 錫=<b 若=<c 寂=<d 弱=<e 惹=<f 主=<g 取=<h
守=<i 手=<j 朱=<k 殊=<l 狩=<m 珠=<n 種=<o 腫=<p 趣=<q 酒=<r
首=<s 儒=<t 受=<u 呪=<v 寿=<w 授=<x 樹=<y 綬=<z 需=<{ 囚=<¦
収=<} 周=<~ 宗==! 就==「 州==# 修==$ 愁==% 拾==& 洲==' 秀==(
秋==) 終==* 繍==+ 習==, 臭==- 舟==. 蒐==/ 衆==0 襲==1 讐==2
蹴==3 輯==4 週==5 酋==6 酬==7 集==8 醜==9 什==: 住==; 充==<
十=== 従==> 戎==? 柔==@ 汁==A 渋==B 獣==C 縦==D 重==E 銃==F
叔==G 夙==H 宿==I 淑==J 祝==K 縮==L 粛==M 塾==N 熟==O 出==P
術==Q 述==R 俊==S 峻==T 春==U 瞬==V 竣==W 舜==X 駿==Y 准==Z
循==〔 旬==¥ 楯==〕 殉==^ 淳==_ 準=== 潤==a 盾==b 純==c 巡==d
遵==e 醇==f 順==g 処==h 初==i 所==j 暑==k 曙==l 渚==m 庶==n
緒==o 署==p 書==q 薯==r 藷==s 諸==t 助==u 叙==v 女==w 序==x
徐==y 恕==z 鋤=={ 除==¦ 傷==} 償==~ 勝=>! 匠=>「 升=># 召=>$
哨=>% 商=>& 唱=>' 嘗=>(奨=>) 妾=>* 娼=>+ 宵=>, 将=>- 小=>.
少=>/ 尚=>0 庄=>1 床=>2 廠=>3 彰=>4 承=>5 抄=>6 招=>7 掌=>8
捷=>9 昇=>: 昌=>; 昭=>< 晶=>= 松=>> 梢=>? 樟=>@ 樵=>A 沼=>B
消=>C 渉=>D 湘=>E 焼=>F 焦=>G 照=>H 症=>I 省=>J 硝=>K 礁=>L
祥=>M 称=>N 章=>O 笑=>P 粧=>Q 紹=>R 肖=>S 菖=>T 蒋=>U 蕉=>V
衝=>W 裳=>X 訟=>Y 証=>Z 詔=>〔 詳=>¥ 象=>〕 賞=>^ 醤=>_ 鉦=>=
鍾=>a 鐘=>b 障=>c 鞘=>d 上=>e 丈=>f 丞=>g 乗=>h 冗=>i 剰=>j

城=>k 場=>l 壌=>m 嬢=>n 常=>o 情=>p 擾=>q 条=>r 杖=>s 浄=>t
状=>u 畳=>v 穣=>w 蒸=>x 譲=>y 醸=>z 錠=>{ 嘱=>¦ 埴=>} 飾=>~
拭=?! 植=?「 殖=?# 燭=?$ 織=?% 職=?& 色=?’ 触=?(食=?) 蝕=?*
辱=?+ 尻=?, 伸=?- 信=?. 侵=?/ 唇=?0 娠=?1 寝=?2 審=?3 心=?4
慎=?5 振=?6 新=?7 晋=?8 森=?9 榛=?: 浸=?; 深=?< 申=?= 疹=?>
真=?? 神=?@ 秦=?A 紳=?B 臣=?C 芯=?D 薪=?E 親=?F 診=?G 身=?H
辛=?I 進=?J 針=?K 震=?L 人=?M 仁=?N 刃=?O 塵=?P 壬=?Q 尋=?R
甚=?S 尽=?T 腎=?U 訊=?V 迅=?W 陣=?X 靭=?Y

〔ス〕 --

笥=?Z 諏=?〔 須=?¥ 酢=?〕 図=?^ 厨=?_ 逗=?= 吹=?a 垂=?b 帥=?c
推=?d 水=?e 炊=?f 睡=?g 粋=?h 翠=?i 衰=?j 遂=?k 酔=?l 錐=?m
錘=?n 随=?o 瑞=?p 髄=?q 崇=?r 嵩=?s 数=?t 枢=?u 趨=?v 雛=?w
据=?x 杉=?y 椙=?z 菅=?{ 頗=?¦ 雀=?} 裾=?~ 澄=@! 摺=@「 寸=@#

〔セ〕 --

世=@$ 瀬=@% 畝=@& 是=@’ 凄=@(制=@) 勢=@* 姓=@+ 征=@, 性=@-
成=@. 政=@/ 整=@0 星=@1 晴=@2 棲=@3 栖=@4 正=@5 清=@6 牲=@7
生=@8 盛=@9 精=@: 聖=@; 声=@< 製=@= 西=@> 誠=@? 誓=@@ 請=@A
逝=@B 醒=@C 青=@D 静=@E 斉=@F 税=@G 脆=@H 隻=@I 席=@J 惜=@K
戚=@L 斥=@M 昔=@N 析=@O 石=@P 積=@Q 籍=@R 績=@S 脊=@T 責=@U
赤=@V 跡=@W 蹟=@X 碩=@Y 切=@Z 拙=@〔 接=@¥ 摂=@〕 折=@^ 設=@_
窃=@= 節=@a 説=@b 雪=@c 絶=@d 舌=@e 蝉=@f 仙=@g 先=@h 千=@i
占=@j 宣=@k 専=@l 尖=@m 川=@n 戦=@o 扇=@p 撰=@q 栓=@r 栴=@s
泉=@t 浅=@u 洗=@v 染=@w 潜=@x 煎=@y 煽=@z 旋=@{ 穿=@¦ 箭=@}
線=@~ 繊=A! 羨=A「 腺=A# 舛=A$ 船=A% 薦=A& 詮=A’ 賎=A(践=A)
選=A* 遷=A+ 銭=A, 銑=A- 閃=A. 鮮=A/ 前=A0 善=A1 漸=A2 然=A3
全=A4 禅=A5 繕=A6 膳=A7 糎=A8

〔ソ〕 --

噌=A9 塑=A: 岨=A; 措=A< 曾=A= 曽=A> 楚=A? 狙=A@ 疏=AA 疎=AB
礎=AC 祖=AD 租=AE 粗=AF 素=AG 組=AH 蘇=AI 訴=AJ 阻=AK 遡=AL
鼠=AM 僧=AN 創=AO 双=AP 叢=AQ 倉=AR 喪=AS 壮=AT 奏=AU 爽=AV
宋=AW 層=AX 匝=AY 惣=AZ 想=A〔 捜=A¥ 掃=A〕 挿=A^ 掻=A_ 操=A=
早=Aa 曹=Ab 巣=Ac 槍=Ad 槽=Ae 漕=Af 燥=Ag 争=Ah 痩=Ai 相=Aj
窓=Ak 糟=Al 総=Am 綜=An 聡=Ao 草=Ap 荘=Aq 葬=Ar 蒼=As 藻=At
装=Au 走=Av 送=Aw 遭=Ax 鎗=Ay 霜=Az 騒=A{ 像=A¦ 増=A} 憎=A~
臓=B! 蔵=B「 贈=B# 造=B$ 促=B% 側=B& 則=B’ 即=B(息=B) 捉=B*
束=B+ 測=B, 足=B- 速=B. 俗=B/ 属=B0 賊=B1 族=B2 続=B3 卒=B4
袖=B5 其=B6 揃=B7 存=B8 孫=B9 尊=B: 損=B; 村=B< 遜=B=

〔タ〕 --

他=B> 多=B? 太=B@ 汰=BA 詑=BB 唾=BC 堕=BD 妥=BE 惰=BF 打=BG
柁=BH 舵=BI 楕=BJ 陀=BK 駄=BL 騨=BM 体=BN 堆=BO 対=BP 耐=BQ
岱=BR 帯=BS 待=BT 怠=BU 態=BV 戴=BW 替=BX 泰=BY 滞=BZ 胎=B〔
腿=B¥ 苔=B〕 袋=B^ 貸=B_ 退=B= 逮=Ba 隊=Bb 黛=Bc 鯛=Bd 代=Be
台=Bf 大=Bg 第=Bh 醍=Bi 題=Bj 鷹=Bk 滝=Bl 瀧=Bm 卓=Bn 啄=Bo
宅=Bp 托=Bq 択=Br 拓=Bs 沢=Bt 濯=Bu 琢=Bv 託=Bw 鐸=Bx 濁=By
諾=Bz 茸=B{ 凧=B¦ 蛸=B} 只=B~ 叩=C! 但=C「 達=C# 辰=C$ 奪=C%
脱=C& 巽=C’ 竪=C(辿=C) 棚=C* 谷=C+ 狸=C, 鱈=C- 樽=C. 誰=C/
丹=C0 単=C1 嘆=C2 坦=C3 担=C4 探=C5 旦=C6 歎=C7 淡=C8 湛=C9
炭=C: 短=C; 端=C< 箪=C= 綻=C> 耽=C? 胆=C@ 蛋=CA 誕=CB 鍛=CC
団=CD 壇=CE 弾=CF 断=CG 暖=CH 檀=CI 段=CJ 男=CK 談=CL

〔チ〕 --

値=CM 知=CN 地=CO 弛=CP 恥=CQ 智=CR 池=CS 痴=CT 稚=CU 置=CV
致=CW 蜘=CX 遅=CY 馳=CZ 築=C〔 畜=C¥ 竹=C〕 筑=C^ 蓄=C_ 逐=C=
秩=Ca 窒=Cb 茶=Cc 嫡=Cd 着=Ce 中=Cf 仲=Cg 宙=Ch 忠=Ci 抽=Cj
昼=Ck 柱=Cl 注=Cm 虫=Cn 衷=Co 註=Cp 酎=Cq 鋳=Cr 駐=Cs 樗=Ct
瀦=Cu 猪=Cv 苧=Cw 著=Cx 貯=Cy 丁=Cz 兆=C{ 凋=C¦ 喋=C} 寵=C~
帖=D! 帳=D「 庁=D# 弔=D$ 張=D% 彫=D& 徴=D’ 懲=D(挑=D) 暢=D*

朝=D+ 潮=D, 牒=D- 町=D. 眺=D/ 聴=D0 脹=D1 腸=D2 蝶=D3 調=D4
諜=D5 超=D6 跳=D7 銚=D8 長=D9 頂=D: 鳥=D; 勅=D< 捗=D= 直=D>
朕=D? 沈=D@ 珍=DA 賃=DB 鎮=DC 陳=DD

〔ツ〕 --

津=DE 墜=DF 椎=DG 槌=DH 追=DI 鎚=DJ 痛=DK 通=DL 塚=DM 栂=DN
掴=DO 槻=DP 佃=DQ 漬=DR 柘=DS 辻=DT 蔦=DU 綴=DV 鍔=DW 椿=DX
潰=DY 坪=DZ 壷=D(嬬=D¥ 紬=D) 爪=D^ 吊=D_ 釣=D= 鶴=Da

〔テ〕 --

亭=Db 低=Dc 停=Dd 偵=De 剃=Df 貞=Dg 呈=Dh 堤=Di 定=Dj 帝=Dk
底=Dl 庭=Dm 廷=Dn 弟=Do 悌=Dp 抵=Dq 挺=Dr 提=Ds 梯=Dt 汀=Du
碇=Dv 禎=Dw 程=Dx 締=Dy 艇=Dz 訂=D{ 諦=D¦ 蹄=D} 逓=D~ 邸=E!
鄭=E" 釘=E# 鼎=E$ 泥=E% 摘=E& 擢=E' 敵=E(滴=E) 的=E* 笛=E+
適=E, 鏑=E- 溺=E. 哲=E/ 徹=E0 撤=E1 轍=E2 迭=E3 鉄=E4 典=E5
填=E6 天=E7 展=E8 店=E9 添=E: 纏=E; 甜=E< 貼=E= 転=E> 顛=E?
点=E@ 伝=EA 殿=EB 澱=EC 田=ED 電=EE

〔ト〕 --

兎=EF 吐=EG 堵=EH 塗=EI 妬=EJ 屠=EK 徒=EL 斗=EM 杜=EN 渡=EO
登=EP 菟=EQ 賭=ER 途=ES 都=ET 鍍=EU 砥=EV 礪=EW 努=EX 度=EY
土=EZ 奴=E(怒=E¥ 倒=E) 党=E^ 冬=E_ 凍=E= 刀=Ea 唐=Eb 塔=Ec
塘=Ed 套=Ee 宕=Ef 島=Eg 嶋=Eh 悼=Ei 投=Ej 搭=Ek 東=El 桃=Em
檮=En 棟=Eo 盗=Ep 淘=Eq 湯=Er 濤=Es 灯=Et 燈=Eu 当=Ev 痘=Ew
祷=Ex 等=Ey 答=Ez 筒=E{ 糖=E¦ 統=E} 到=E~ 董=F! 蕩=F" 藤=F#
討=F$ 謄=F% 豆=F& 踏=F' 逃=F(透=F) 鐙=F* 陶=F+ 頭=F, 騰=F-
闘=F. 働=F/ 動=F0 同=F1 堂=F2 導=F3 憧=F4 撞=F5 洞=F6 瞳=F7
童=F8 胴=F9 萄=F: 道=F; 銅=F< 峠=F= 鴇=F> 匿=F? 得=F@ 徳=FA
涜=FB 特=FC 督=FD 禿=FE 篤=FF 毒=FG 独=FH 読=FI 栃=FJ 橡=FK
凸=FL 突=FM 椴=FN 届=FO 鳶=FP 苫=FQ 寅=FR 酉=FS 瀞=FT 噸=FU
屯=FV 惇=FW 敦=FX 沌=FY 豚=FZ 遁=F(頓=F¥ 呑=F) 曇=F^ 鈍=F_

〔ナ〕 --

奈=F= 那=Fa 内=Fb 乍=Fc 凪=Fd 薙=Fe 謎=Ff 灘=Fg 捺=Fh 鍋=Fi
楢=Fj 馴=Fk 縄=Fl 畷=Fm 南=Fn 楠=Fo 軟=Fp 難=Fq 汝=Fr

〔ニ〕 --

二=Fs 尼=Ft 弐=Fu 迩=Fv 匂=Fw 賑=Fx 肉=Fy 虹=Fz 廿=F{ 日=F¦
乳=F} 入=F~ 如=G! 尿=G" 韮=G# 任=G$ 妊=G% 忍=G& 認=G'

〔ヌ〕 --

濡=G(

〔ネ〕 --

禰=G) 祢=G* 寧=G+ 葱=G, 猫=G- 熱=G. 年=G/ 念=G0 捻=G1 撚=G2
燃=G3 粘=G4

〔ノ〕 --

乃=G5 廼=G6 之=G7 埜=G8 嚢=G9 悩=G: 濃=G; 納=G< 能=G= 脳=G>
膿=G? 農=G@ 覗=GA 蚤=GB

〔ハ〕 --

巴=GC 把=GD 播=GE 覇=GF 杷=GG 波=GH 派=GI 琶=GJ 破=GK 婆=GL
罵=GM 芭=GN 馬=GO 俳=GP 廃=GQ 拝=GR 排=GS 敗=GT 杯=GU 盃=GV
牌=GW 背=GX 肺=GY 輩=GZ 配=G(倍=G¥ 培=G) 媒=G^ 梅=G_ 楳=G=
煤=Ga 狽=Gb 買=Gc 売=Gd 賠=Ge 陪=Gf 這=Gg 蝿=Gh 秤=Gi 矧=Gj
萩=Gk 伯=Gl 剥=Gm 博=Gn 拍=Go 柏=Gp 泊=Gq 白=Gr 箔=Gs 粕=Gt
舶=Gu 薄=Gv 迫=Gw 曝=Gx 漠=Gy 爆=Gz 縛=G{ 莫=G¦ 駁=G} 麦=G~
函=H! 箱=H" 硲=H# 箸=H$ 肇=H% 筈=H& 櫨=H' 幡=H(肌=H) 畑=H*
畠=H+ 八=H, 鉢=H- 溌=H. 発=H/ 醗=H0 髪=H1 伐=H2 罰=H3 抜=H4
筏=H5 閥=H6 鳩=H7 噺=H8 塙=H9 蛤=H: 隼=H; 伴=H< 判=H= 半=H>
反=H? 叛=H@ 帆=HA 搬=HB 斑=HC 板=HD 氾=HE 汎=HF 版=HG 犯=HH
班=HI 畔=HJ 繁=HK 般=HL 藩=HM 販=HN 範=HO 釆=HP 煩=HQ 頒=HR
飯=HS 挽=HT 晩=HU 番=HV 盤=HW 磐=HX 蕃=HY 蛮=HZ

〔ヒ〕 --

匪=H〔 卑=H¥ 否=H〕 妃=H^ 庇=H_ 彼=H= 悲=Ha 扉=Hb 批=Hc 披=Hd
斐=He 比=Hf 泌=Hg 疲=Hh 皮=Hi 碑=Hj 秘=Hk 緋=Hl 罷=Hm 肥=Hn
被=Ho 誹=Hp 費=Hq 避=Hr 非=Hs 飛=Ht 樋=Hu 簸=Hv 備=Hw 尾=Hx
微=Hy 枇=Hz 毘=H{ 琵=H¦ 眉=H} 美=H~ 鼻=I! 柊=I「 稗=I# 匹=I$
疋=I% 髭=I& 彦=I’ 膝=I(菱=I) 肘=I* 弼=I+ 必=I, 畢=I- 筆=I.
逼=I/ 桧=I0 姫=I1 媛=I2 紐=I3 百=I4 謬=I5 俵=I6 彪=I7 標=I8
氷=I9 漂=I: 瓢=I; 票=I< 表=I= 評=I> 豹=I? 廟=I@ 描=IA 病=IB
秒=IC 苗=ID 錨=IE 鋲=IF 蒜=IG 蛭=IH 鰭=II 品=IJ 彬=IK 斌=IL
浜=IM 瀕=IN 貧=IO 賓=IP 頻=IQ 敏=IR 瓶=IS

〔フ〕 --

不=IT 付=IU 埠=IV 夫=IW 婦=IX 富=IY 冨=IZ 布=I〔 府=I¥ 怖=I〕
扶=I^ 敷=I_ 斧=I= 普=Ia 浮=Ib 父=Ic 符=Id 腐=Ie 膚=If 芙=Ig
譜=Ih 負=Ii 賦=Ij 赴=Ik 阜=Il 附=Im 侮=In 撫=Io 武=Ip 舞=Iq
葡=Ir 蕪=Is 部=It 封=Iu 楓=Iv 風=Iw 葺=Ix 蕗=Iy 伏=Iz 副=I{
復=I¦ 幅=I} 服=I~ 福=J! 腹=J「 複=J# 覆=J$ 淵=J% 弗=J& 払=J’
沸=J(仏=J) 物=J* 鮒=J+ 分=J, 吻=J- 噴=J. 墳=J/ 憤=J0 扮=J1
焚=J2 奮=J3 粉=J4 糞=J5 紛=J6 雰=J7 文=J8 聞=J9

〔ヘ〕 --

丙=J: 併=J; 兵=J< 塀=J= 幣=J> 平=J? 弊=J@ 柄=JA 並=JB 蔽=JC
閉=JD 陛=JE 米=JF 頁=JG 僻=JH 壁=JI 癖=JJ 碧=JK 別=JL 瞥=JM
蔑=JN 箆=JO 偏=JP 変=JQ 片=JR 篇=JS 編=JT 辺=JU 返=JV 遍=JW
便=JX 勉=JY 娩=JZ 弁=J〔 鞭=J¥

〔ホ〕 --

保=J〕 舗=J^ 鋪=J_ 圃=J= 捕=Ja 歩=Jb 甫=Jc 補=Jd 輔=Je 穂=Jf
募=Jg 墓=Jh 慕=Ji 戊=Jj 暮=Jk 母=Jl 簿=Jm 菩=Jn 倣=Jo 俸=Jp
包=Jq 呆=Jr 報=Js 奉=Jt 宝=Ju 峰=Jv 峯=Jw 崩=Jx 庖=Jy 抱=Jz
捧=J{ 放=J¦ 方=J} 朋=J~ 法=K! 泡=K「 烹=K# 砲=K$ 縫=K% 胞=K&
芳=K’ 萌=K(蓬=K) 蜂=K* 褒=K+ 訪=K, 豊=K- 邦=K. 鋒=K/ 飽=K0
鳳=K1 鵬=K2 乏=K3 亡=K4 傍=K5 剖=K6 坊=K7 妨=K8 帽=K9 忘=K:
忙=K; 房=K< 暴=K= 望=K> 某=K? 棒=K@ 冒=KA 紡=KB 肪=KC 膨=KD
謀=KE 貌=KF 貿=KG 鉾=KH 防=KI 吠=KJ 頬=KK 北=KL 僕=KM 卜=KN
墨=KO 撲=KP 朴=KQ 牧=KR 睦=KS 穆=KT 釦=KU 勃=KV 没=KW 殆=KX
堀=KY 幌=KZ 奔=K〔 本=K¥ 翻=K〕 凡=K^ 盆=K_

〔マ〕 --

摩=K= 磨=Ka 魔=Kb 麻=Kc 埋=Kd 妹=Ke 昧=Kf 枚=Kg 毎=Kh 哩=Ki
槙=Kj 幕=Kk 膜=Kl 枕=Km 鮪=Kn 柾=Ko 鱒=Kp 桝=Kq 亦=Kr 俣=Ks
又=Kt 抹=Ku 末=Kv 沫=Kw 迄=Kx 儘=Ky 繭=Kz 麿=K{ 万=K¦ 慢=K}
満=K~ 漫=L! 蔓=L「

〔ミ〕 --

味=L# 未=L$ 魅=L% 巳=L& 箕=L’ 岬=L(密=L) 蜜=L* 湊=L+ 蓑=L,
稔=L- 脈=L. 妙=L/ 粍=L0 民=L1 眠=L2

〔ム〕 --

務=L3 夢=L4 無=L5 牟=L6 矛=L7 霧=L8 鵡=L9 椋=L: 婿=L; 娘=L<

〔メ〕 --

冥=L= 名=L> 命=L? 明=L@ 盟=LA 迷=LB 銘=LC 鳴=LD 姪=LE 牝=LF
滅=LG 免=LH 棉=LI 綿=LJ 緬=LK 面=LL 麺=LM

〔モ〕 --

摸=LN 模=LO 茂=LP 妄=LQ 孟=LR 毛=LS 猛=LT 盲=LU 網=LV 耗=LW
蒙=LX 儲=LY 木=LZ 黙=L〔 目=L¥ 杢=L〕 勿=L^ 餅=L_ 尤=L= 戻=La
籾=Lb 貰=Lc 問=Ld 悶=Le 紋=Lf 門=Lg 匁=Lh

〔ヤ〕 --

也=Li 冶=Lj 夜=Lk 爺=Ll 耶=Lm 野=Ln 弥=Lo 矢=Lp 厄=Lq 役=Lr
約=Ls 薬=Lt 訳=Lu 躍=Lv 靖=Lw 柳=Lx 薮=Ly 鑓=Lz

〔ユ〕 --

愉=L{ 愈=L¦ 油=L} 癒=L~ 諭=M! 輸=M「 唯=M# 佑=M$ 優=M% 勇=M&

友=M’ 宥=M（ 幽=M） 悠=M＊ 憂=M+ 揖=M， 有=M－ 柚=M． 湧=M／ 涌=M0
猶=M1 猷=M2 由=M3 祐=M4 裕=M5 誘=M6 遊=M7 邑=M8 郵=M9 雄=M：
融=M； 夕=M<

〔ヨ〕 --

予=M= 余=M> 与=M? 誉=M@ 輿=MA 預=MB 傭=MC 幼=MD 妖=ME 容=MF
庸=MG 揚=MH 揺=MI 擁=MJ 曜=MK 楊=ML 様=MM 洋=MN 溶=MO 熔=MP
用=MQ 窯=MR 羊=MS 耀=MT 葉=MU 蓉=MV 要=MW 謡=MX 踊=MY 遥=MZ
陽=M〔 養=M¥ 慾=M〕 抑=M^ 欲=M＿ 沃=M= 浴=Ma 翌=Mb 翼=Mc 淀=Md

〔ラ〕 --

羅=Me 螺=Mf 裸=Mg 来=Mh 莱=Mi 頼=Mj 雷=Mk 洛=Ml 絡=Mm 落=Mn
酪=Mo 乱=Mp 卵=Mq 嵐=Mr 欄=Ms 濫=Mt 藍=Mu 蘭=Mv 覧=Mw

〔リ〕 --

利=Mx 吏=My 履=Mz 李=M{ 梨=M¦ 理=M} 璃=M~ 痢=N! 裏=N「 裡=N#
里=N$ 離=N% 陸=N& 律=N’ 率=N（ 立=N） 葎=N＊ 掠=N+ 略=N， 劉=N－
流=N． 溜=N／ 琉=N0 留=N1 硫=N2 粒=N3 隆=N4 竜=N5 龍=N6 侶=N7
慮=N8 旅=N9 虜=N： 了=N； 亮=N< 僚=N= 両=N> 凌=N? 寮=N@ 料=NA
梁=NB 涼=NC 猟=ND 療=NE 瞭=NF 稜=NG 糧=NH 良=NI 諒=NJ 遼=NK
量=NL 陵=NM 領=NN 力=NO 緑=NP 倫=NQ 厘=NR 林=NS 淋=NT 燐=NU
琳=NV 臨=NW 輪=NX 隣=NY 鱗=NZ 麟=N〔

〔ル〕 --

瑠=N¥ 塁=N〕 涙=N^ 累=N＿ 類=N=

〔レ〕 --

令=Na 伶=Nb 例=Nc 冷=Nd 励=Ne 嶺=Nf 怜=Ng 玲=Nh 礼=Ni 苓=Nj
鈴=Nk 隷=Nl 零=Nm 霊=Nn 麗=No 齢=Np 暦=Nq 歴=Nr 列=Ns 劣=Nt
烈=Nu 裂=Nv 廉=Nw 恋=Nx 憐=Ny 漣=Nz 煉=N{ 簾=N¦ 練=N} 聯=N~
蓮=O! 連=O「 錬=O#

〔ロ〕 --

呂=O$ 魯=O% 櫓=O& 炉=O’ 賂=O（ 路=O） 露=O＊ 労=O+ 婁=O， 廊=O－
弄=O． 朗=O／ 楼=O0 榔=O1 浪=O2 漏=O3 牢=O4 狼=O5 籠=O6 老=O7
聾=O8 蝋=O9 郎=O： 六=O； 麓=O< 禄=O= 肋=O> 録=O? 論=O@

〔ワ〕 --

倭=OA 和=OB 話=OC 歪=OD 賄=OE 脇=OF 惑=OG 枠=OH 鷲=OI 亙=OJ
亘=OK 鰐=OL 詫=OM 藁=ON 蕨=OO 椀=OP 湾=OQ 碗=OR 腕=OS

付録－ 9　プリンタ機能一覧表

分　　類	ニーモニック	HEXコード	機　　　能	PC-8023(C)	NK-3618 PC-8821/22	PC-PR201	NM-9300(S) NM-9400(S)
印字指令	CR	0D	バッファのデータを印字	○	○	○	○
改　　行	LF	0A	1行送り	○	○	○	○
垂直タブ	VT	0B	多行送り	○	○	○	○
フォームフィード	FF	0C	改ページ	○	○	○	○
拡　　大	SO	0E	拡大指令（8bit）	○	○	○	○
（8bit）	SI	0F	拡大解除（8bit）	○	○	○	○
セレクト	DC1	11	セレクト	○	○	○	○
ディセレクト	DC3	13	ディセレクト	○	○	○	○
拡　　大	DC2	12	拡大指令（7bit）	○	○	○	○
（7bit）	DC4	14	拡大解除（7bit）	○	○	○	○
水平タブ	HT	09	水平タブ移動	○	○	○	○
キャンセル	CAN	18	データのキャンセル	○	○	○	○
n行改行	US	1F	1～15行の改行	○	○	○	○
V F U	—	—	タブ位置等の設定	○	○	○	○
印字方法	ESC, N	1B, 4E	HSパイカ	○	○	○	○
	ESC, P	1B, 50	プロポーショナル	○	○	○	○
	ESC, Q	1B, 51	コンデンス	○	○	○	○
	ESC, E	1B, 45	エリート	○	○	○	○
	ESC, H	1B, 48	HDパイカ	×	○	○	○
	ESC, K	1B, 4B	漢字（横）	×	○	○	○
	ESC, t	1B, 74	漢字（縦）	×	×	○	○
ドットスペース	ESC, SOH	1B, 01	1ドットスペース	○	○	○	○
	ESC, STX	1B, 02	2ドットスペース	○	○	○	○
	ESC, ETX	1B, 03	3ドットスペース	○	○	○	○
	ESC, EOT	1B, 04	4ドットスペース	○	○	○	○
	ESC, ENQ	1B, 05	5ドットスペース	○	○	○	○
	ESC, ACK	1B, 06	6ドットスペース	○	○	○	○
	ESC, BEL	1B, 07	7ドットスペース	×	×	○	○
	ESC, BS	1B, 08	8ドットスペース	×	×	○	○
キャラクタモード	ESC, $	1B, 24	英数記号モード	○	○	○	○
	ESC, &	1B, 26	ひらがなモード	○	○	○	○
	ESC, ＃	1B, 23	内部グラフィックモード	×	○	○	○
ドット列印字モード	ESC, S	1B, 53	8bit ドット列	○	○	○	○
	ESC, I	1B, 49	16bit ドット列	×	○	○	○
	ESC, J	1B, 4A	24bit ドット列	×	×	○	○
	ESC, V	1B, 56	8bit ドット列リピート	×	○	○	○
	ESC, W	1B, 57	16bit ドット列リピート	×	○	○	○
	ESC, U	1B, 55	24bit ドット列リピート	×	×	○	○

分　類	ニーモニック	HEXコード	機　能	PC-8023(C)	NK-3618 PC-8821/22	PC-PR201	NM-9300(S) NM-9400(S)
	ESC, F	1B, 46	ドットアドレッシング	×	○	○	○
キャラクタリピート	ESC, R	1B, 52	キャラクタリピート	×	○	○	○
強調文字	ESC, !	1B, 21	強調文字セレクト	○	○	○	○
	ESC, ”	1B, 22	強調文字解除	○	○	○	○
印字モード	ESC,]	1B, 5D	ロジカルシークモード	○	○	○	○
	ESC, ＞	1B, 3E	片方向印字	×	○	○	○
	ESC, [	1B, 5B	インクリメンタルモード	○	×	×	×
改行幅	ESC, A	1B, 41	1/6インチ改行モード	○	○	○	○
	ESC, B	1B, 42	1/8インチ改行モード	○	○	○	○
	ESC, T	1B, 54	N/120インチ改行モード	○	○	○	○
			N/160インチ改行モード	×	×	×	○*1
改行方向	ESC, f	1B, 66	順方向改行モード	○	○	○	○
	ESC, r	1B, 72	逆方向改行モード	○	○	○	○
水平タブ	ESC, (	1B, 28	水平タブセット	○	○	○	○
	ESC,)	1B, 29	水平タブ部分クリア	○	○	○	○
	ESC, 2	1B, 30	水平タブオールクリア	○	○	○	○
アンダーライン	ESC, X	1B, 58	アンダーライン開始	○	○	○	○
	ESC, Y	1B, 59	アンダーライン終了	○	○	○	○
レフトマージン	ESC, L	1B, 4C	印字開始位置の設定	○	○	○	○
ライトマージン	ESC, ／	1B, 2F	印字終了位置の設定	×	×	○	○
リボン切換	ESC, C	1B, 43	リボン切換指定	×	○	×	○
外字のロード	ESC, *	1B, 2A	外字のロード(16×16ドット)	×	○	○	○
	ESC, +	1B, 2B	外字のロード(24×24ドット)	×	×	○	○
ドット対応グラフィックドット数の切り換え	ESC, D	1B, 44	640ドットモード（コピーモード）	×	○	○	○
	ESC, M	1B, 4D	960ドットモード（ネイティブモード）	×	○	○	○
シフトフィーダ	ESC, a	1B, 61	全排出後全吸入	×	×	○	×
	ESC, b	1B, 62	全排出	×	×	○	×

*1）NM-9300S/9400SではN/180インチ改行

NM-9300/9400 CEX シーケンス（NM-9300S/9400SではSUBシーケンス）

分　　類	ニーモニックコード	HEXコード	機　　　能
ドット列印字モード	CEX, A	1A, 41	960ドット/8インチ基本ピッチ
	CEX, B	1A, 42	1280ドット/8インチ基本ピッチ
	CEX, C	1A, 43	1440ドット/8インチ基本ピッチ
改行基本ピッチ	CEX, F	1A, 47	1/120インチ改行ピッチ
	CEX, G	1A, 46	1/160インチ改行ピッチ *1
外字ロードクリア	CEX, 1	1A, 31	外字登録部分クリア
	CEX, 2	1A, 32	外字登録オールクリア
スーパ/サブスクリプトモード	CEX, U	1A, 55	スーパースクリプトモード
	CEX, L	1A, 4C	サブスクリプトモード
文字の縦拡大	CEX, V	1A, 56	縦拡大文字モード
	CEX, W	1A, 57	縦拡大の解除
漢字モードピッチ選択	CEX, Q	1A, 51	漢字24ドットピッチ
	CEX, N	1A, 4E	漢字27ドットピッチ
	CEX, E	1A, 45	漢字30ドットピッチ
	CEX, P	1A, 50	漢字36ドットピッチ

付録-10　機械語オペレーション一覧表

(1) イミディエート値

n　：　8ビット

nn：　16ビット

(2) フラグ

●　＝　変化しない

○　＝　リセットされる

1　＝　セットされる

×　＝　不　定

↕　＝　操作の結果でセット/リセットされる

IFF　＝　割り込みイネーブル・フリップ・フロップの状態

P　＝　P/Vフラグはパリティフラグとして扱われる

V　＝　P/Vフラグはオーバーフロー・フラグとして扱われる

8ビット・ロード

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備考
			C	Z	P/V	S	N	H		
MOV r,r'	LD r,r'	r←r'	●	●	●	●	●	●	4	
MVI r,n	LD r,n	r←n	●	●	●	●	●	●	7	
MOV r,M	LD r,(HL)	r←(HL)	●	●	●	●	●	●	7	
	LD r,(IX+d)	r←(IX+d)	●	●	●	●	●	●	19	
	LD r,(IY+d)	r←(IY+d)	●	●	●	●	●	●	19	
MOV M,r	LD (HL),r	(HL)←r	●	●	●	●	●	●	7	
	LD (IX+d),r	(IX+d)←r	●	●	●	●	●	●	19	
	LD (IY+d),r	(IY+d)←r	●	●	●	●	●	●	19	
MVI M,n	LD (HL),n	(HL)←n	●	●	●	●	●	●	10	
	LD (IX+d),n	(IX+d)←n	●	●	●	●	●	●	19	
	LD (IY+d),n	(IY+d)←n	●	●	●	●	●	●	19	
LDAX B	LD A,(BC)	A←(BC)	●	●	●	●	●	●	7	
LDAX D	LD A,(DE)	A←(DE)	●	●	●	●	●	●	7	
LDA nn	LD A,(nn)	A←(nn)	●	●	●	●	●	●	13	
STAX B	LD (BC),A	(BC)←A	●	●	●	●	●	●	7	
STAX D	LD (DE),A	(DE)←A	●	●	●	●	●	●	7	
STA nn	LD (nn),A	(nn)←A	●	●	●	●	●	●	13	
	LD A,I	A←I	●	↕	IFF	↕	0	0	9	
	LD A,R	A←R	●	↕	IFF	↕	0	0	9	
	LD I,A	I←A	●	●	●	●	●	●	9	
	LD R,A	R←A	●	●	●	●	●	●	9	

注) r,r'はA, B, C, D, E, H, Lレジスタを指す。
IFF(割り込みイネーブル・フリップ・フロップ)はP/Vフラグにコピーされる。

8ビット算術・論理演算

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備考
			C	Z	P/V	S	N	H		
ADD r	ADD A,r	A←A+r	↕	↕	V	↕	0	↕	4	
ADI n	ADD A,n	A←A+n	↕	↕	V	↕	0	↕	7	
ADD M	ADD A,(HL)	A←A+(HL)	↕	↕	V	↕	0	↕	7	
	ADD A,(IX+d)	A←A+(IX+d)	↕	↕	V	↕	0	↕	19	
	ADD A,(IY+d)	A←A+(IY+d)	↕	↕	V	↕	0	↕	19	
ADC S	ADC A,s	A←A+s+CY	↕	↕	V	↕	0	↕	ADDと同じ	
SUB S	SUB s	A←A−s	↕	↕	V	↕	1	↕		
SBB S	SBC A,s	A←A−s−CY	↕	↕	V	↕	1	↕		
ANI ,1 ANA S	AND s	A←A∧s	0	↕	P	↕	0	↕		
ORI ,1 ORA S	OR s	A←A∨s	0	↕	P	↕	0	↕		
XRI ,1 XRA S	XOR s	A←Axs	0	↕	P	↕	0	↕		
CPI ,1 CMPS	CP s	A−s	↕	↕	V	↕	1	↕		
INR r	INC r	r←r+1	●	↕	V	↕	0	↕	4	
INR M	INC (HL)	(HL)←(HL)+1	●	↕	V	↕	0	↕	11	
	INC (IX+d)	(IX+d)← (IX+d)+1	●	↕	V	↕	0	↕	23	
	INC (IY+d)	(IY+d)← (IY+d)+1	●	↕	V	↕	0	↕	23	
DCR M DCR r	DEC m	m←m−1	●	↕	V	↕	1	↕	INCと同じ	

Sはr,n,(HL),(IX+d),(IY+d)を表す。
mはr,(HL),(IX+d),(IY+d)を表す。

16ビット・ロード

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備　考
			C	Z	P/V	S	N	H		
LXI dd,nn	LD dd,nn	dd←nn	●	●	●	●	●	●	10	
	LD IX,nn	IX←nn	●	●	●	●	●	●	14	
	LD IY,nn	IY←nn	●	●	●	●	●	●	14	
LHLD nn	LD HL,(nn)	H←(nn+1) L←(nn)	●	●	●	●	●	●	16	
	LD dd,(nn)	dd_H←(nn+1) dd_L+(nn)	●	●	●	●	●	●	20	
	LD IX, (nn)	IX_H←(nn+1) IX_L←(nn)	●	●	●	●	●	●	20	
	LD IY, (nn)	IY_H←(nn+1) IY_L←(nn)	●	●	●	●	●	●	16	
SHLD nn	LD(nn),HL	(nn+1)←H (nn)←L	●	●	●	●	●	●	20	
	LD(nn),dd	(nn+1)←dd_H (nn)←dd_L	●	●	●	●	●	●	20	nnは2バイト数 下位1バイトは0P コードの直後。 上位1バイトは その次に入る。
	LD(nn),IX	(nn+1)←IX_H (nn)←IX_L	●	●	●	●	●	●	20	
	LD(nn),IY	(nn+1)←IY_H (nn)←IY_L	●	●	●	●	●	●	20	
SPHL	LD SP,HL	SP←HL	●	●	●	●	●	●	6	
	LD SP,IX	SP←IX	●	●	●	●	●	●	10	
	LD SP,IY	SP←IY	●	●	●	●	●	●	10	
PUSH qq	PUSH qq'	(SP-2)←qq_L (SP-1)←qq_H	●	●	●	●	●	●	11	
	PUSH IX	(SP-2)←IX_L (SP-1)←IX_H	●	●	●	●	●	●	15	
	PUSH IY	(SP-2)←IY_L (SP-1)←IY_H	●	●	●	●	●	●	15	
POP qq	POP qq	qq_H←(SP+1) qq_L←(SP)	●	●	●	●	●	●	10	
	POP IX	IX_H←(SP+1) IX_L←(SP)	●	●	●	●	●	●	14	
	POP IY	IY_H←(SP+1) IY_L←(SP)	●	●	●	●	●	●	14	

注）ddはレジスタ・ペアBC, DE, HL, SP。
qqはレジスタ・ペアAF, BC, DE, HL。
添字H, Lはそれぞれ高位バイト、低位バイトを表わす。
例）BC_L=C, AF_H=A

16ビット算術演算

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備　　考
			S	C	Z	P/V	S	N		
DAD ss	ADD HL,ss	HL←HL+ss	↕	●	●	●	0	X	11	
	ADC HL,ss	HL←HL+ss+CY	↕	↕	V	↕	0	X	15	
	SBC HL,ss	HL←HL−ss−CY	↕	↕	V	↕	1	X	15	
	ADD IX,pp	IX←IX+pp	↕	●	●	●	0	X	15	
	ADD IY,rr	IY←IY+rr	↕	●	●	●	0	X	15	
INX ss	INC ss	ss←ss+1	●	●	●	●	●	●	6	
	INC IX	IX←IX+1	●	●	●	●	●	●	10	
	INC IY	IY←IY+1	●	●	●	●	●	●	10	
DCX ss	DEC,ss	ss←ss−1	●	●	●	●	●	●	6	
	DEC IX	IX←IX−1	●	●	●	●	●	●	10	
	DEC IY	IY←IY−1	●	●	●	●	●	●	10	

注）ssはレジスタ・ペアBC, DE, HL, SP。
　　ppはレジスタ・ペアBC, DE, IX, SP。
　　rrはレジスタ・ペアBC、DE、IY、SP。

ジャンプ

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化 C	 Z	 P/V	 S	 N	 H	実行時間	備　　考
JMP nn	JP nn	PC←nn	●	●	●	●	●	●	10	
JNZ JZ JNC JC JPO JPE JP JM } nn	JP cc, nn	ccが真ならば PC←nn, その他は次へ	●	●	●	●	●	●	10	
JR e	JR e	PC←PC+e	●	●	●	●	●	●	12	
JRC e*	JR C, e	C=1ならば PC←PC+e	●	●	●	●	●	●	12	CC:条件　フラグ NZ:ノンゼロ (Z=0)　(Z=0) Z :ゼロ (Z=1)　(Z=1) NC:ノンキャリー (C=0)　(C=0) C :キャリー (C=1)　(C=1) PO:パリティ奇数 (P/V=0)　(P/V=0) PE:パリティ偶数 (P/V=1)　(P/V=1) P :正 (S=0)　(S=0) M :負 (S=1)　(S=1)
		C=0ならば 次へ							7	
JRNC e*	JR NC, e	C=0ならば PC←PC+e	●	●	●	●	●	●	12	
		C=1ならば 次へ							7	
JRZ e*	JR Z, e	Z=1ならば PC←PC+e	●	●	●	●	●	●	12	
		Z=0ならば 次へ							7	
JRNZ e*	JR NZ, e	Z=0ならば PC←PC+e	●	●	●	●	●	●	12	
		Z=1ならば 次へ							7	
PCHL	JP(HL)	PC←HL	●	●	●	●	●	●	4	
	JP(IX)	PC←IX	●	●	●	●	●	●	8	
	JP(IY)	PC←IY	●	●	●	●	●	●	8	
DJNZ e*	DJNZ e	B←B−1 B≒0ならば PC←PC+e	●	●	●	●	●	●	13	
		B=0ならば 次へ							8	

注）eは相封アドレシンダ・モードでのイクステンション
　　eは符号付は2の補数値(−126～+129)。
　　eそのものは、OPコードの位置から計算した値である。
　　・はPC-8801mkIIモニタでのみ使用可。

コール、リターン

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化 C	Z	P/V	S	N	H	実行時間	備考
CALL nn	CALL nn	(SP−1)←PC_H (SP−2)←PC_L PC←nn	●	●	●	●	●	●	17	
CNZ CZ CNC CC CPO OPE CP CM } nn	CALL cc, nn	ccが真ならば CALL nnと同じ	●	●	●	●	●	●	17	
		その他ならば 次へ							10	
RET	RET	PC_L←(SP) PC_H←(SP+1)	●	●	●	●	●	●	10	
RNZ RZ RNL RC RPO RPE RP RM	RET cc	ccが真ならば RETと同じ	●	●	●	●	●	●	11	
		その他ならば 次へ							5	
	RETI	割り込みからの リターン	●	●	●	●	●	●	14	
	RETN	ノン・マスカブル 割り込みからの リターン	●	●	●	●	●	●	14	
RST p RST 0 ～ 7	RST p	(SP−1)←PC_H (SP−2)←PC_L PC_H←0 PC_L←p	●	●	●	●	●	●	11	

備考:

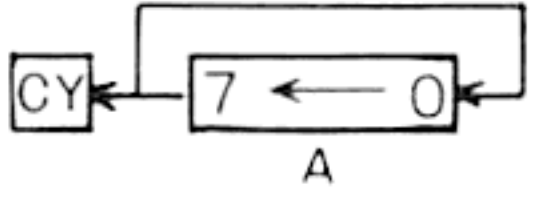
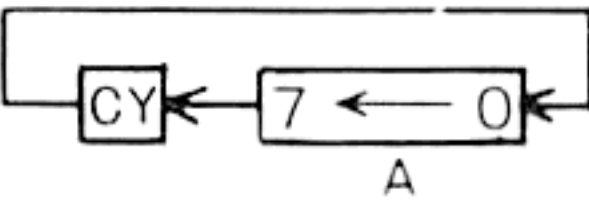
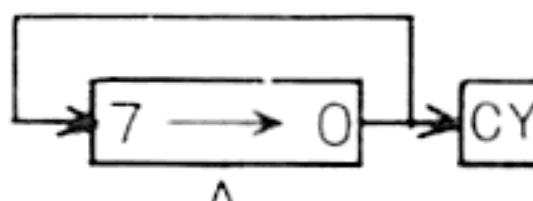
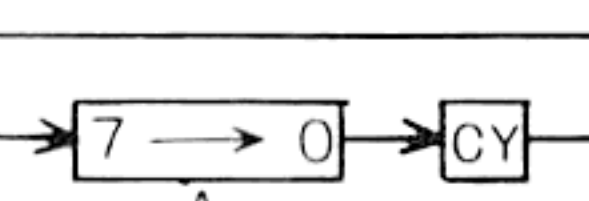
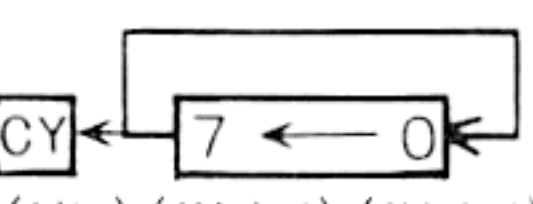
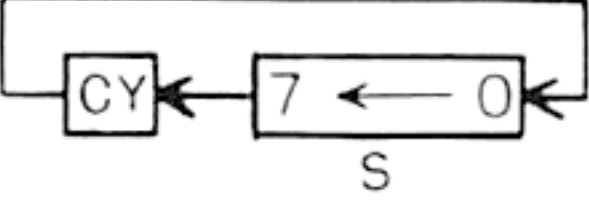
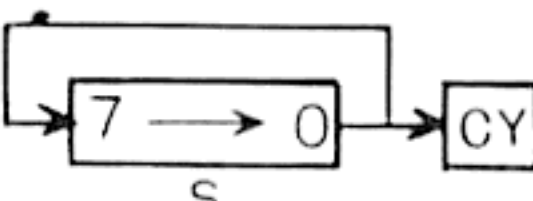
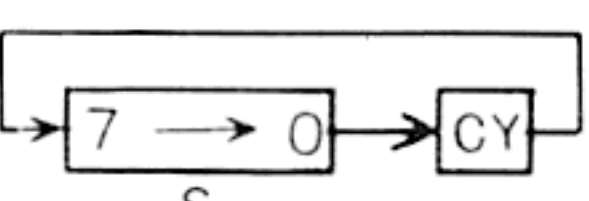
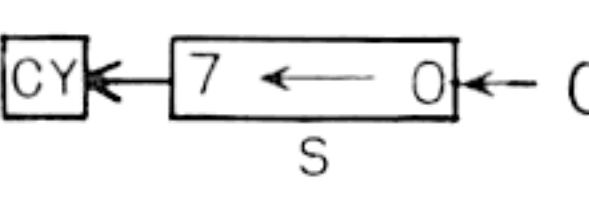
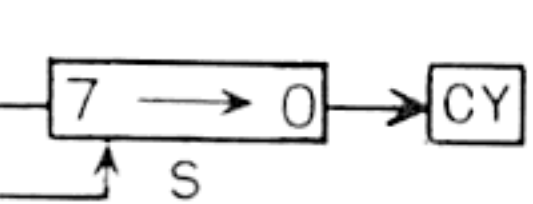
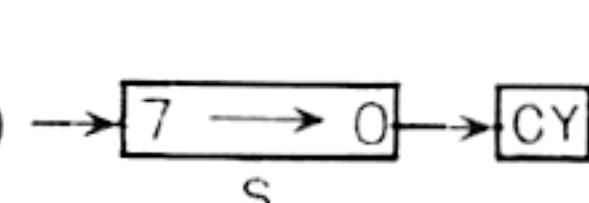
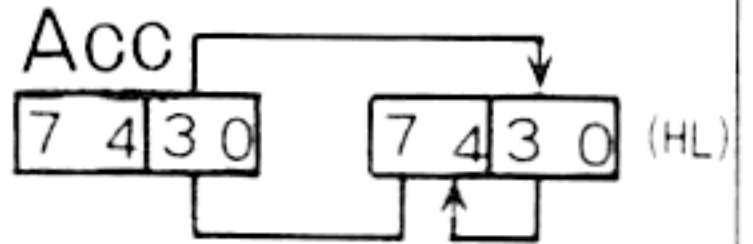
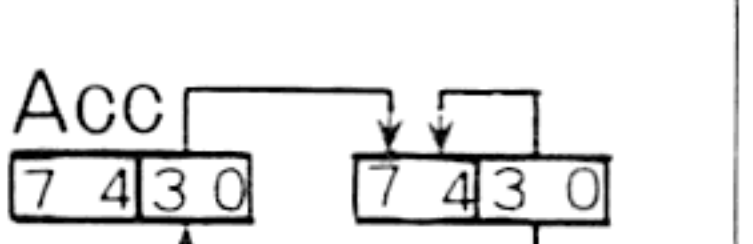
CC:条件	フラグ
NZ:ノンゼロ	(Z=0)
Z :ゼロ	(Z=1)
NC:ノンキャリー	(C=0)
C :キャリー	(C=1)
PO:パリティ奇数	(P/V=0)
PE:パリティ偶数	(P/V=1)
P :正	(S=0)
M :負	(S=1)

交換、ブロック転送・サーチ

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備　　考
			C	Z	P/V	S	N	H		
XCHG	EX DE, HL	DE↔HL	●	●	●	●	●	●	4	
	EX AF, AF'	AF↔AF'	●	●	●	●	●	●	4	
	EXX	(BC, DE, HL) ↔ (BC', DE', HL')	●	●	●	●	●	●	4	レジスタの切換
XTHL	EX (SP),HL	H↔(SP+1) L↔(SP)	●	●	●	●	●	●	19	
	EX (SP),IX	IX_H↔(SP+1) IX_L↔(SP)	●	●	●	●	●	●	23	
	EX (SP),IY	IY_H↔(SP+1) IY_L↔(SP)	●	●	●	●	●	●	23	
	LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	●	●	① ↕	●	0	0	16	ポインタ 1増 バイト・カウンタ 1減
	LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	●	●	0	●	0	0	21	BC≒0のとき
		BC=0ならば 終わり							16	BC=0のとき
	LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	●	●	① ↕	●	0	0	16	
	LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	●	●	0	●	0	0	21	BC≒0のとき
		BC=0ならば 終わり							16	BC=0のとき
	CPI	A−(HL) HL←HL+1 BC←BC−1	●	② ↕	① ↕	↕	1	↕	16	
	CPIR	A−(HL) HL←HL+1 BC←BC−1	●	② ↕	① ↕	↕	1	↕	21	BC≒0かつ A≒(HL)のとき
		A=(HL)または BC=0ならば 終わり							16	BC=0か A=(HL)のとき
	CPD	A−(HL) HL←HL−1 BC←BC−1	●	② ↕	① ↕	↕	1	↕	16	
	CPDR	A−(HL) HL←HL−1 BC←BC−1	●	② ↕	① ↕	↕	1	↕	21	BC≒0かつ A≒(HL)のとき
		A=(HL)または BC=0ならば 終わり							16	BC=0か A=(HL)のとき

注）①BC−1=0ならばP/Vは0、その他は1。
　　②A=(HL)ならばZは1、その他は0。

各種操作およびＣＰＵ制御

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備　　考
			C	Z	P/V	S	N	H		
DAA	DAA	10進補正 （加算、減算）	↕	↕	P	↕	●	↕	4	デシマル・アジャスト ・アキュムレータ
CMA	CPL	$A \leftarrow \bar{A}$	●	●	●	●	1	1	4	1 の補数
	NEG	A←0←A	↕	↕	V	↕	1	↕	8	2 の補数
CMC	CCF	$CY \leftarrow \overline{CY}$	↕	●	●	●	0	×	4	キャリの反転
STC	SCF	CY←1	1	●	●	●	0	0	4	キャリ・セット
NOP	NOP	No operation	●	●	●	●	●	●	4	
HLT	HALT	CPU待機	●	●	●	●	●	●	4	
	DI	IFF←0	●	●	●	●	●	●	4	ディスエーブル割り込み
	EI	IFF←1	●	●	●	●	●	●	4	イネーブル割り込み
	IM0	MODE 0 にセット	●	●	●	●	●	●	8	割り込みモードのセット
	IM1	MODE 1 にセット	●	●	●	●	●	●	8	
	IM2	MODE 2 にセット	●	●	●	●	●	●	8	

注）IFFは割り込みフリップ・フロップ。
　　CYはキャリフラグ。

入力／出力

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化 C	Z	P/V	S	N	H	実行時間	備考
IN n	IN A, (n)	A←(n)	●	●	●	●	●	●	11	n←A_0~A_7 Acc←A_0~A_{15}
	INr, (C)	r←(C)	●	↕	P	↕	0	↕	12	
	INI	(HL)←(C) B←B−1 HL←HL+1	●	① ↕	× ×	×	1	×	16	C←A_0~A_7 B→A_8~A_{15}
	INIR	(HL)←(C) B←B−1 HL←HL+1 B=0まで繰り返す	●	1	×	×	1	×	21 16	B≠0 のとき B=0 のとき
	IND	(HL)←(C) B←B−1 HL←HL−1	●	① ↕	×	×	1	×	16	
	INDR	(HL)←(C) B←B−1 HL←HL−1 B=0まで繰り返す	●	1	×	×	1	×	21 16	B≠0 のとき B=0 のとき
OUT n	OUT (n),A	(n)←A	●	●	●	●	●	●	11	n→A_0~A_7 Acc→A_5~A_{15}
	OUT (C),r	(C)←r	●	●	●	●	●	●	12	
	OUTI	B←B−1 (C)←(HL) HL←HL+1	●	① ↕	×	×	1	×	16	C→A_0~A_7 B→A_8~A_{15}
	OTIR	B←B−1 (C)←(HL) HL←HL+1 B=0まで繰り返す	●	1	×	×	1	×	21 16	B≠0 のとき B=0 のとき
	OUTD	B←B−1 (C)←(HL) HL←HL−1	●	① ↕	×	×	1	×	16	
	OTDR	B←B−1 (C)←(HL) HL←HL−1 B=0まで繰り返す	●	1	×	×	1	×	21 16	B≠0 のとき B=0 のとき

注）① B−1が0になればZフラグがセットされ、それ以外のときはリセットされる。
② INI, INIR, IND, INDRではデクリメントされる前のBの値がA_8~A_{15}にのる。
③ OUTI, OTIR, OUTD, OTDRではデクリメントされた後のBの値がA_8~A_{15}にのる。

ローテイト、シフト

インテル ニーモニック	ザイログ ニーモニック	シンボリック オペレーション	フラグ変化 C	Z	P/V	S	N	H	実行時間	備　　考
RLC	RLCA	CY ← 7 ← 0 (A)	↕	●	●	●	0	0	4	
RAL	RLA	CY ← 7 ← 0 (A)	↕	●	●	●	0	0	4	
RRC	RRCA	7 → 0 → CY (A)	↕	●	●	●	0	0	4	
RAR	RRA	7 → 0 → CY (A)	↕	●	●	●	0	0	4	
	RLC r	CY ← 7 ← 0 r,(HL),(IX+d),(IY+d)	↕	↕	P	↕	0	0	8	
	RLC (HL)		↕	↕	P	↕	0	0	15	
	RLCI (X+d)		↕	↕	P	↕	0	0	23	
	RLCI (Y+d)		↕	↕	P	↕	0	0	23	
	RL s	CY ← 7 ← 0 (s)	↕	↕	P	↕	0	0	RLC と同じ	
	RRC s	7 → 0 → CY (s)	↕	↕	P	↕	0	0		
	RR s	7 → 0 → CY (s)	↕	↕	P	↕	0	0		
	SLA s	CY ← 7 ← 0 ← 0 (s)	↕	↕	P	↕	0	0		
	SRA s	7 → 0 → CY (s)	↕	↕	P	↕	0	0		
	SRL s	0 → 7 → 0 → CY (s)	↕	↕	P	↕	0	0		
	RLD	Acc 7 4 3 0　7 4 3 0 (HL)	●	↕	P	↕	0	0	18	
	RRD	Acc 7 4 3 0　7 4 3 0	●	↕	P	↕	0	0	18	

S は r, (HL), (IX+d), (IY+d) を表す。

ビット操作、テスト

インテル ニーモニック	ザイロク ニーモニック	シンボリック オペレーション	フラグ変化						実行時間	備考
			C	Z	P/V	S	N	H		
	BIT b, r	Z←$\bar{r}$b	●	↕	×	×	0	1	8	
	BIT b, (HL)	Z←(HL)b	●	↕	×	×	0	1	12	
	BIT b, (IX+d)	Z←(IX+d)b	●	↕	×	×	0	1	20	
	BIT b, (IY+d)	Z←(IY+d)b	●	↕	×	×	0	1	20	
	SET b, r	rb←1	●	●	●	●	●	●	8	
	SET b, (HL)	(HL)b←1	●	●	●	●	●	●	15	
	SET b, (IX+d)	(IX+d)b←1	●	●	●	●	●	●	23	
	SET b, (IY+d)	(IY+d)b←1	●	●	●	●	●	●	23	
	RES b, m	Sb←0 m≡r, (HL), (IX+d) (IY+d)	●	●	●	●	●	●	SET と同じ	

付-11 16進コード→ニーモニック早見表

→上位4ビット

	0	1	2	3	4	5	6	7	
0	NOP	DJNZ b	JR NZ,b	JR NC,b	LD B,B	LD D,B	LD H,B	LD (HL),B	0
1	LD BC,b_3b_2	LD DE,b_3b_2	LD HL,b_3b_2	LD SP,b_3b_2	LD B,C	LD D,C	LD H,C	LD (HL),C	1
2	LD (BC),A	LD (DE),A	LD (b_3b_2),HL	LD (b_3b_2),A	LD B,D	LD D,D	LD H,D	LD (HL),D	2
3	INC BC	INC DE	INC HL	INC SP	LD B,E	LD D,E	LD H,E	LD (HL),E	3
4	INC B	INC D	INC H	INC (HL)	LD B,H	LD D,H	LD H,H	LD (HL),H	4
5	DEC B	DEC D	DEC H	DEC (HL)	LD B,L	LD D,L	LD H,L	LD (HL),L	5
6	LD B,b_2	LD D,b_2	LD H,b_2	LD (HL),b_2	LD B,(HL)	LD D,(HL)	LD H,(HL)	HALT	6
7	RLCA	RLA	DAA	SCF	LD B,A	LD D,A	LD H,A	LD (HL),A	7
8	EX AF,A'F'	JR b_2	JR Z,b_2	JR C,b_2	LD C,B	LD E,B	LD L,B	LD A,B	8
9	ADD HL,BC	ADD HL,DE	ADD HL,HL	ADD HL,SP	LD C,C	LD E,C	LD L,C	LD A,C	9
A	LD A,(BC)	LD A,(DE)	LD HL,b_3b_2	LD A,(b_3b_2)	LD C,D	LD E,D	LD L,D	LD A,D	A
B	DEC BC	DEC DE	DEC HL	DEC SP	LD C,E	LD E,E	LD L,E	LD A,E	B
C	INC C	INC E	INC L	INC A	LD C,H	LD E,H	LD L,H	LD A,H	C
D	DEC C	DEC E	DEC L	DEC A	LD C,L	LD E,L	LD L,L	LD A,L	D
E	LD C,b_2	LD E,b_2	LD L,b_2	LD A,b_2	LD C,(HL)	LD E,(HL)	LD L,(HL)	LD A,(HL)	E
F	RRCA	RRA	CPL	CCF	LD C,A	LD E,A	LD L,A	LD A,A	F

①CB＋

	0	1	2	3	4	5	6	7
0	RLC B	RL B	SLA B		BIT 0,B	BIT 2,B	BIT 4,B	BIT 6,B
1	RLC C	RL C	SLA C		BIT 0,C	BIT 2,C	BIT 4,C	BIT 6,C
2	RLC D	RL D	SLA D		BIT 0,D	BIT 2,D	BIT 4,D	BIT 6,D
3	RLC E	RL E	SLA E		BIT 0,E	BIT 2,E	BIT 4,E	BIT 6,E
4	RLC H	RL H	SLA H		BIT 0,H	BIT 2,H	BIT 4,H	BIT 6,H
5	RLC L	RL L	SLA L		BIT 0,L	BIT 2,L	BIT 4,L	BIT 6,L
6	RLC (HL)	RL (HL)	SLA (HL)		BIT 0,(HL)	BIT 2,(HL)	BIT 4,(HL)	BIT 6,(HL)
7	RLC A	RL A	SLA A		BIT 0,A	BIT 2,A	BIT 4,A	BIT 6,A
8	RRC B	RR B	SRA B	SRL B	BIT 1,B	BIT 3,B	BIT 5,B	BIT 7,B
9	RRC C	RR C	SRA C	SRL C	BIT 1,C	BIT 3,C	BIT 5,C	BIT 7,C
A	RRC D	RR D	SRA D	SRL D	BIT 1,D	BIT 3,D	BIT 5,D	BIT 7,D
B	RRC E	RR E	SRA E	SRL E	BIT 1,E	BIT 3,E	BIT 5,E	BIT 7,E
C	RRC H	RR H	SRA H	SRL H	BIT 1,H	BIT 3,H	BIT 5,H	BIT 7,H
D	RRC L	RR L	SRA L	SRL L	BIT 1,L	BIT 3,L	BIT 5,L	BIT 7,L
E	RRC (HL)	RR (HL)	SRA (HL)	SRL (HL)	BIT 1,(HL)	BIT 3,(HL)	BIT 5,(HL)	BIT 7,(HL)
F	RRC A	RR A	SRA A	SRL A	BIT 1,A	BIT 3,A	BIT 5,A	BIT 7,A

②DD＋④FD＋(表中のIXをIYに)

09	ADD IX,BC	66	LD H,(IX+b_3)	B6	OR (IX+b_3)
19	ADD IX,DE	6E	LD L,(IX+b_3)	BE	CP (IX+b_3)
21	LD IX,b_4b_3	70	LD (IX+b_3),B	CB	*
22	LD (b_4b_3),IX	71	LD (IX+b_3),C	E1	POP IX
23	INC IX	72	LD (IX+b_3),D	E3	EX (SP),IX
29	ADD IX,IX	73	LD (IX+b_3),E	E5	PUSH IX
2A	LD IX,(b_4b_3)	74	LD (IX+b_3),H	E9	JP (IX)
2B	DEC IX	75	LD (IX+b_3),L	F9	LD SP,IX
34	INC (IX+b_3)	77	LD (IX+b_3),A		
35	DEC (IX+b_3)	7E	LD A,(IX+b_3)		
36	LD (IX+b_3),b_4	86	ADD A,(IX+b_3)		
39	ADD IX,SP	8E	ADC A,(IX+b_3)		
46	LD B,(IX+b_3)	96	SUB (IX+b_3)		
5E	LD C,(IX+b_3)	9E	SBC A,(IX+b_3)		
56	LD D,(IX+b_3)	A6	AND (IX+b_3)		
5E	LD E,(IX+b_3)	AE	XOR (IX+b_3)		

DD·CB＋(FD·CB＋)

06	RLC (IX+b_3)	86	RES 0,(IX+b_3)
0E	RRC (IX+b_3)	8E	RES 1,(IX+b_3)
16	RL (IX+b_3)	96	RES 2,(IX+b_3)
1E	RR (IX+b_3)	9E	RES 3,(IX+b_3)
26	SLA (IX+b_3)	A6	RES 4,(IX+b_3)
2E	SRA (IX+b_3)	AE	RES 5,(IX+b_3)
3E	SRL (IX+b_3)	B6	RES 6,(IX+b_3)
		BE	RES 7,(IX+b_3)
46	BIT 0,(IX+b_3)	C6	SET 0,(IX+b_3)
4E	BIT 1,(IX+b_3)	CE	SET 1,(IX+b_3)
56	BIT 2,(IX+b_3)	D6	SET 2,(IX+b_3)
5E	BIT 3,(IX+b_3)	DE	SET 3,(IX+b_3)
66	BIT 4,(IX+b_3)	E6	SET 4,(IX+b_3)
6E	BIT 5,(IX+b_3)	EE	SET 5,(IX+b_3)
76	BIT 6,(IX+b_3)	F6	SET 6,(IX+b_3)
7E	BIT 7,(IX+b_3)	FE	SET 7,(IX+b_3)

	8	9	A	B	C	D	E	F	
0	ADD A,B	SUB B	AND B	OR B	RET NZ	RET NC	RET PO	RET P	0
1	ADD A,C	SUB C	AND C	OR C	POP BC	POP DE	POP HL	POP AF	1
2	ADD A,D	SUB D	AND D	OR D	JP NZ,b_3b_2	JP NC,b_3b_2	JP PO,b_3b_2	JP P,b_3b_2	2
3	ADD A,E	SUB E	AND E	OR E	JP b_3b_2	OUT (b_2),A	EX (SP),HL	DI	3
4	ADD A,H	SUB H	AND H	OR H	CALL NZ,b_3b_2	CALL NC,b_3b_2	CALL PO,b_3b_2	CALL P,b_3b_2	4
5	ADD A,L	SUB L	AND L	OR L	PUSH BC	PUSH DE	PUSH HL	PUSH AF	5
6	ADD A,(HL)	SUB (HL)	AND (HL)	OR (HL)	ADD A,b_2	SUB b_2	AND b_2	OR b_2	6
7	ADD A,A	SUB A	AND A	OR A	RST 0H	RST 10H	RST 20H	RST 30H	7
8	ADD A,B	SBC B	XOR B	CP B	RET Z	RET C	RET PE	RET M	8
9	ADD A,C	SBC C	XOR C	CP C	RET	EXX	JP (HL)	LD SP,HL	9
A	ADD A,D	SBC D	XOR D	CP D	JP Z,b_3b_2	JP C,b_3b_2	JP PE,b_3b_2	JP M,b_3b_2	A
B	ADD A,E	SBC E	XOR E	CP E	①	IN A,(b_2)	EX DE,HL	EI	B
C	ADC A,H	SBC H	XOR H	CP H	CALL Z,b_3b_2	CALL C,b_3b_2	CALL PE,b_3b_2	CALL M,b_3b_2	C
D	ADC A,L	SBC L	XOR L	CP L	CALL b_3b_2	②	③	④	D
E	ADC A,(HL)	SBC (HL)	XOR (HL)	CP (HL)	ADC A,b_2	SBC b_2	XOR b_2	CP b_2	E
F	ADC A,A	SBC A	XOR A	CP A	RST 8H	RST 18H	RST 28H	RST 38H	F

	8	9	A	B	C	D	E	F
0	RES 0,B	RES 2,B	RES 4,B	RES 6,B	SET 0,B	SET 2,B	SET 4,B	SET 6,B
1	RES 0,C	RES 2,C	RES 4,C	RES 6,C	SET 0,C	SET 2,C	SET 4,C	SET 6,C
2	RES 0,D	RES 2,D	RES 4,D	RES 6,D	SET 0,D	SET 2,D	SET 4,D	SET 6,D
3	RES 0,E	RES 2,E	RES 4,E	RES 6,E	SET 0,E	SET 2,E	SET 4,E	SET 6,E
4	RES 0,H	RES 2,H	RES 4,H	RES 6,H	SET 0,H	SET 2,H	SET 4,H	SET 6,H
5	RES 0,L	RES 2,L	RES 4,L	RES 6,L	SET 0,L	SET 2,L	SET 4,L	SET 6,L
6	RES 0,(HL)	RES 2,(HL)	RES 4,(HL)	RES 6,(HL)	SET 0,(HL)	SET 2,(HL)	SET 4,(HL)	SET 6,(HL)
7	RES 0,A	RES 2,A	RES 4,A	RES 6,A	SET 0,A	SET 2,A	SET 4,A	SET 6,A
8	RES 1,B	RES 3,B	RES 5,B	RES 7,B	SET 1,B	SET 3,B	SET 5,B	SET 7,B
9	RES 1,C	RES 3,C	RES 5,C	RES 7,C	SET 1,C	SET 3,C	SET 5,C	SET 7,C
A	RES 1,D	RES 3,D	RES 5,D	RES 7,D	SET 1,D	SET 3,D	SET 5,D	SET 7,D
B	RES 1,E	RES 3,E	RES 5,E	RES 7,E	SET 1,E	SET 3,E	SET 5,E	SET 7,E
C	RES 1,H	RES 3,H	RES 5,H	RES 7,H	SET 1,H	SET 3,H	SET 5,H	SET 7,H
D	RES 1,L	RES 3,L	RES 5,L	RES 7,L	SET 1,L	SET 3,L	SET 5,L	SET 7,L
E	RES 1,(HL)	RES 3,(HL)	RES 5,(HL)	RES 7,(HL)	SET 1,(HL)	SET 3,(HL)	SET 5,(HL)	SET 7,(HL)
F	RES 1,A	RES 3,A	RES 5,A	RES 7,A	SET 1,A	SET 3,A	SET 5,A	SET 7,A

③ ED＋

	4	5	6	7	A	B
0	IN B,(C)	IN D,(C)	IN H,(C)		LDI	LDIR
1	OUT (C),B	OUT (C),D	OUT (C),H		CPI	CPIR
2	SBC HL,BC	SBC HL,DE	SBC HL,HL	SBC HL,SP	INI	INIR
3	LD (b_4b_3),BC	LD (b_4b_3),DE	LD (b_3b_2),HL	LD (b_4b_3),SP	OUTI	OTIR
4	NEG					
5	RETN					
6	IM 0	IM 1				
7	LD I,A	LD AJ				
8	IN C,(C)	IN E,(C)	IN L,(C)	IN A,(C)	LDD	LDDR
9	OUT C,(C)	OUT (C),E	OUT (C),L	OUT (C),A	CPD	CPDR
A	ADC HL,BC	ADC HL,DE	ADC HL,HL	ADC HL,SP	IND	INDR
B	LD BC,(b_4b_3)	LD DE,(b_4b_3)	LD HL,(b_4b_3)	LD SP,(b_4b_3)	OUTD	OTDR
C						
D	RETI					
E		IM 2				
F	LD R,A	LD A,R	RLD			

付-12 ニーモニック→16進コード早見表

8ビットロード命令

	B	C	D	E	H	L	(HL)	A	n	(IX+n)	(IY+n)
LD B	40	41	42	43	44	45	46	47	06	DD·46	FD·46
C	48	49	4A	4B	4C	4D	4E	4F	0E	DD·4E	FD·4E
D	50	51	52	53	54	55	56	57	16	DD·56	FD·56
E	58	59	5A	5B	5C	5D	5E	5F	1E	DD·5E	FD·5E
H	60	61	62	63	64	65	66	67	26	DD·66	FD·66
L	68	69	6A	6B	6C	6D	6E	6F	2E	DD·6E	FD·6E
(HL)	70	71	72	73	74	75	76	77	36	DD·76	FD·76
A	78	79	7A	7B	7C	7D	7E	7F	3E	DD·7E	FD·7E

	A
LD I	ED·47
R	ED·4E
(BC)	02
(DE)	12
(nn)	32

	I	R	(BC)	(DE)	(nn)
LD A	ED·57	ED·5F	0A	1A	3A

16ビットロード命令

	nn	(nn)
LD BC	01	ED·4B
DE	11	ED·5B
HL	21	2A
SP	31	ED·7B
IX	DD·21	DD·2A
IY	FD·21	FD·2A

	BC	DE	HL	IX	IY	SP
LD SP			F9	DD·F9	FD·F9	
(nn)	ED·43	ED·53	22	DD·22	FD·22	ED·73

8ビット算術演算命令

	B	C	D	E	H	L	(HL)	A	(IX+n)	(IY+n)	n
ADD A	80	81	82	83	84	85	86	87	DD·86	FD·86	C6
ADC A	88	89	8A	8B	8C	8D	8E	8F	DD·8E	FD·8E	CE
SUB	90	91	92	93	94	95	96	97	DD·96	FD·96	D6
SBC	98	99	9A	9B	9C	9D	9E	9F	DD·9E	FD·9E	DE
AND	A0	A1	A2	A3	A4	A5	A6	A7	DD·A6	FD·A6	E6
XOR	A8	A9	AA	AB	AC	AD	AE	AF	DD·AE	FD·AE	EE
OR	B0	B1	B2	B3	B4	B5	B6	B7	DD·B6	FD·B6	F6
CP	B8	B9	BA	BB	BC	BD	BE	BF	DD·BE	FD·BE	FE
INC	04	0C	14	1C	24	2C	34	3C	DD·34	FD·34	
DEC	05	0D	15	1D	25	2D	35	3D	DD·35	FD·35	

16ビット算術演算命令

	BC	DE	HL	SP
ADD HL	09	19	29	39
ADC HL	ED·4A	ED·5A	ED·6A	ED·7A
SBC HL	ED·42	ED·52	ED·62	ED·72
INC	03	13	23	33
DEC	0B	1B	2B	3B

	BC	CE	HL	SP
ADD IX	DD·09	DD·19	DD·29	DD·39
ADD IY	FD·09	FD·19	FD·29	FD·39

レジスタ交換命令

EX DE, HL	EB
EX AF, AF	08
EXX	D9

EX(SP), HL	E3
EX(SP), IX	DD·E3
EX(SP), IY	FD·E3

ブロック転送・比較・入出力命令(ED+)

	I	IR	D	DR	動作	
LD	A0	B0	A8	B8	(DE)←(HL) BC←BC−1	DE←DE±1 HL←HL±1
CP	A1	B1	A9	B9	A−(HL)	BC←BC−1 HL←HL±1
IN	A2	B2	AA	BA	(HL)←(C)	B←B−1 HL←HL±1
OUT	A3	B3	AB	BB	(C)←(HL)	B←B−1 HL←HL±1

ビット操作命令(CB＋)（ＩX, ＩYのDD, FDはCBの前につける）

	B	C	D	E	H	L	(HL)	A	(IX+n)	(IY+n)
BIT 0	40	41	42	43	44	45	46	47	DD·46	FD·46
1	48	49	4A	4B	4C	4D	4E	4F	DD·4E	FD·4E
2	50	51	52	53	54	55	56	57	DD·56	FD·56
3	58	59	5A	5B	5C	5D	5E	5F	DD·5E	FD·5E
4	60	61	62	63	64	65	66	67	DD·66	FD·66
5	68	69	6A	6B	6C	6D	6E	6F	DD·6E	FD·6E
6	70	71	72	73	74	75	76	77	DD·76	FD·76
7	78	79	7A	7B	7C	7D	7E	7F	DD·7E	FD·7E

RES
8X
8X
9X
9X
AX
AX
BX
BX

SET
CX
CX
DX
DX
EX
EX
FX
FX

回転・シフト命令(CB＋)……RLCA, RRCA, RLA, RRAは"CB"不要

	B	C	D	E	H	L	(HL)	A	(IX+n)	(IY+n)	動　作
RLC	00	01	02	03	04	05	06	07	DD·06	FD·06	
RRC	08	09	0A	0B	0C	0D	0E	0F	DD·0E	FD·0E	
RL	10	11	12	13	14	15	16	17	DD·16	FD·16	
RR	18	19	1A	1B	1C	1D	1E	1F	DD·1E	FD·1E	
SLA	20	21	22	23	24	25	26	27	DD·26	FD·26	0
SRA	28	29	2A	2B	2C	2D	2E	2F	DD·2E	FD·2E	
SRL	38	39	3A	3B	3C	3D	3E	3F	DD·3E	FD·3E	0

RRD	ED·67
ACC……	
(HL)……	

RLD	ED·6F
ACC……	
(HL)……	

レジスタ退避・解除命令

	BC	DE	HL	AF	IX	IY
PUSH	C5	D5	E5	F5	DD·E5	FD·E5
POP	C1	D1	E1	F1	DD·E1	FD·E1

ジャンプ・コール・リターン命令

		Z	NZ	C	NC	PE	PO	P	M
JR	18	28	20	38	30				
JP	C3	CA	C2	DA	D2	EA	E2	F2	FA
CALL	CD	CC	C4	DC	D4	EC	E4	F4	FC
RET	C9	C8	C0	D8	D0	E8	E0	F0	F8

JP (HL)	E9
(IX)	DD·E9
(IY)	FD·E9
DJNZ	10
RETI	ED·4D
RETN	ED·45

入出力命令

IN A, (n)	DB
OUT (n), A	D3

(ED＋)

	B	C	D	E	H	L	A
IN r, (C)	40	48	50	58	60	68	78
OUT (C), r	41	49	51	59	61	69	79

CPUコントロール

EI	FB
DI	F3
IM 0	ED·46
IM 1	ED·56
IM 2	ED·5E
NOP	00
HALT	76

リスタート

RST 0H	C7
8H	CF
10H	D7
18H	DF
20H	E7
28H	EF
30H	F7
38H	FF

AFレジスタ操作

DAA	27
CPL	2F
NEG	ED·44
CCF	3F
SCF	37

—索　　引—

昭和60年4月25日　第1版第1刷発行
定価　2,900円

著　者　平松　達雄　八木　良一
発行者　樺島　正博
発行所　株式会社システムソフト
〒810　福岡市中央区渡辺通2-4-8
代表電話　092-714-6236
郵便振替　福岡3-37311
印 刷 所　大日本印刷

ISBN4-88235-013-0　C0055　¥2900E